

IEEE Standard for Information Technology Portable Operating System Interface (POSIX®)

Base Specifications, Issue 7

IEEE Computer Society

and

The Open Group

Sponsored by the
Portable Applications Standards Committee



IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1003.1 -2017
(Revision of IEEE Std 1003.1-2008)

The Open Group Standard Base Specifications, Issue 7

IEEE Std 1003.1™-2017
(Revision of IEEE Std 1003.1-2008)

The Open Group Standard, Base Specifications, Issue 7

IEEE Standard for Information Technology— Portable Operating System Interface (POSIX®)

Base Specifications, Issue 7

Sponsor

Portable Applications Standards Committee

of the

IEEE Computer Society

and

The Open Group

Abstract

POSIX.1-2017 is simultaneously IEEE Std 1003.1™-2017 and The Open Group Standard Base Specifications, Issue 7.

POSIX.1-2017 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. POSIX.1-2017 is intended to be used by both application developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of POSIX.1-2017 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of POSIX.1-2017:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2017 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, IEEE 1003.1™, input/output (I/O), job control, network, parent, portable operating system interface (POSIX®), shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

IEEE Std 1003.1™-2017 (Revision of IEEE Std 1003.1-2008)
IEEE Standard for Information Technology—Portable Operating System Interface (POSIX®)
Base Specifications, Issue 7

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

The Open Group
Apex Plaza, Forbury Road, Reading, Berkshire RG1 1AX, UK

Copyright © 2018 by The Institute of Electrical and Electronics Engineers, Inc. and The Open Group.
All rights reserved.

Published 31 January 2018 by the IEEE.

PDF: ISBN 978-1-5044-4542-9 STD22909
Print: ISBN 978-1-5044-4543-6 STDPD22909

Published 31 January 2018 by The Open Group.

Doc. Number: C181
ISBN: 1-947754-05-08

IEEE is a registered trademark in the US Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

This release of the standard is dedicated to the memory of Roger Faulkner.

POSIX is a registered trademark of IEEE.

This standard has been prepared by the Austin Group. Feedback relating to the material contained within this standard may be submitted by using the Austin Group web site at www.opengroup.org/austin/defectform.html.

IEEE prohibits discrimination, harassment, and bullying. For more information, visit www.ieee.org/web/aboutus/whatis/policies/p9-26.html.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher. Permission to reproduce all or any part of this standard must be with the consent of both copyright holders and may be subject to a license fee. Both copyright holders will need to be satisfied that the other has granted permission. Requests should be sent by email to austin-group-permissions@opengroup.org.

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards. Our diverse membership of more than 550 organizations includes customers, systems and solutions suppliers, tools vendors, integrators, academics, and consultants across multiple industries.

The Open Group aims to:

- Capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Open Group Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at <http://www.opengroup.org/library>.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Standards Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied, and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE Standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE

UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official Statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Xplore Website at <http://ieeexplore.ieee.org/> or contact IEEE at the address listed previously. For more information about the IEEE SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

IEEE Std 1003.1™-2017 was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/IEC JTC 1/SC22.

The Austin Group

At the time this IEEE standard was completed, the Austin Group had the following membership:

Andrew Josey, Chair
Donald W. Cragun, Organizational Representative, IEEE PASC
Nicholas M. Stoughton, Organizational Representative, ISO/IEC JTC 1/SC22
Martin Řehák, Organizational Representative, The Open Group
Cathy Fox, Technical Editor

Austin Group Technical Reviewers

Bogdan Barbu	Aurelio Jargas	Torvald Riegel
Eric Blake	Andrew Josey	Xavier Roche
Harvey Block	John Kearney	Askar Safin
Mark S. Brown	Michael Kerrisk	Anton Salikhmetov
Milan Cermak	Rob King	Jörg Schilling
Stephane Chazelas	Andi Kleen	Ed Schouten
Alexander Cherepanov	Bruce Korb	Konrad Schwarz
Mark Clancy	Kaz Kylheku	Jens Schweikhardt
Geoff Clare	Antoine Leca	Mike Scudder
David Clissold	Wojtek Lerch	Martin Sebor
Alan Coopersmith	Sven Mascheck	Nicholas M. Stoughton
Donald W. Cragun	Margot Hackett Miller	Keith Thompson
Matthew Dempsky	Joseph S. Myers	Jilles Tjoelker
Dan Douglas	Szabolcs Nagy	William Toth
Lawrence D.K.B. Dwyer	Alexander Nasonov	Daniel Trebbien
Paul Eggert	Jonathan Nieder	Miloslav Trmac
Steve Emmerson	Danny Niu	Fred J. Tydeman
David Egan Evans	Gian Ntzik	Ted Unangst
Roger Faulkner	Steffen Nurpmeso	Brian Utterback
Richard Felker	Carlos O'Donell	Stijn van Drongelen
Thorsten Glaser	Vladimir Támara Patiño	Jonathan Wakely
Glenn D. Golden	Peter Petrov	Nathan Weeks
Jim Grisanzio	William Pitcock	Florian Weimer
Philip Guenther	Jim Pryor	David A. Wheeler
Richard Hansen	James C. Pugsley	Michael Wilson
Mark Harris	Yury Pukhalsky	André Zepezauer
Cyril Hrubis	Steve Rago	Mark Ziegast
Jarmo Jaakkola	Chet Ramey	
Felix Janda	Martin Řehák	

Austin Group Working Group Members

Eitan Adler	Philip Guenther	James C. Pugsley
Iwan Aucamp	Joseph M. Gwinn	Yury Pukhalsky
Bogdan Barbu	Bruno Haible	Steve Rago
David Bartley	Richard Hansen	Chet Ramey
Steve Bartolomei	Mark Harris	Martin Řehák
Fabrice Bauzac	Barry E. Hedquist	Tom Ridge

Guido Berhoerster	David Holland	Torvald Riegel
Eric Blake	Tom Honermann	Xavier Roche
Harvey Block	Cyril Hrubis	Askar Safin
Hans Boehm	Jarmo Jaakkola	Anton Salikhmetov
Bill Bogstad	Felix Janda	Jörg Schilling
Pádraig Brady	Aurelio Jargas	Ed Schouten
Davide Brini	Ross Johnson	Konrad Schwarz
Andries E. Brouwer	Andrew Josey	Ingo Schwarze
Mark S. Brown	John Kearney	Jens Schweikhardt
David Butenhof	Dan Kegel	Mike Scudder
Milan Cermak	Michael Kerrisk	Martin Sebor
Stephane Chazelas	Rob King	Glen Seeds
Alexander Cherepanov	Tomas Klacko	Jeffrey Sheinberg
Mark Clancy	Andi Kleen	Thor Lancelot Simon
Geoff Clare	Bruce Korb	Keld Simonsen
David Clissold	David Korn	Ranjit Singh
Alan Coopersmith	Kaz Kylheku	Paul Smith
Donald W. Cragun	Rob Landley	Steven Stewart-Gallus
Martijn Dekker	Antoine Leca	Nicholas M. Stoughton
Matthew Dempsky	Vincent Lefèvre	Alfred M. Szmidt
Thomas E. Dickey	Sean Leonard	Marcel Telka
Casper Dik	Wojtek Lerch	Donn Terry
Dan Douglas	Yonggang Luo	Keith Thompson
Niall Douglas	Scott Lurndal	Jilles Tjoelker
Ulrich Drepper	Roger Marquis	William Toth
Lawrence D.K.B. Dwyer	Sven Mascheck	Daniel Trebbien
Paul Eggert	Per Mildner	Miloslav Trmac
Daniel Eischen	Margot Hackett Miller	Fred J. Tydeman
Robert Elz	Joseph S. Myers	Ted Unangst
Steve Emmerson	Szabolcs Nagy	Brian Utterback
Laszlo Ersek	Alexander Nasonov	Stijn van Drongelen
Bruce Evans	Jonathan Nieder	Christopher Vance
David Egan Evans	Danny Niu	Jonathan Wakely
Roger Faulkner	Gian Ntzik	Nathan Weeks
Richard Felker	Steffen Nurpmeso	Florian Weimer
Jeffrey K. Fellin	Carlos O'Donnell	David A. Wheeler
Hal Finkel	Isabella Parakiss	Mats D. Wichmann
Glenn Fowler	Vladimir Támara Patiño	Michael Wilson
Cathy Fox	Phil Pennock	Garrett Wollman
Mike Frysinger	Andres Perera	Jörg Wunsch
Andrea Giugliano	Peter Petrov	James Youngman
Thorsten Glaser	William Pitcock	André Zepezauer
Glenn D. Golden	Wayne Pollock	Mark Ziegast
Jim Grisanzio	Jim Pryor	

The Open Group

When The Open Group approved the Base Specifications, Issue 7, 2016 edition (technically identical to this standard) on 20 June 2016, the membership of The Open Group Base Working Group was as follows:

Andrew Josey, Chair
Roger Faulkner, Austin Group Liaison
Cathy Fox, Technical Editor

Base Working Group Members

Eric Blake	Donald W. Cragun	Andrew Josey
Geoff Clare	Roger Faulkner	Martin Řehák
David Clissold	Darrin Johnson	S. R. Srinivasan

IEEE

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the Portable Applications Standards Committee had the following membership:

Joseph M. Gwinn, Chair
Andrew Josey, Functional Chair (Interpretations)
Donald W. Cragun, Shell and Utilities Working Group Chair
Barry Hedquist, Test Methods Working Group Chair
Craig Meyers, Distributed Services Working Group Chair
Stephen Walli, US TAG Institutional Representative
Roger Martin, Ex-officio Emeritus
Nicholas M. Stoughton, Secretary

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Charles Barest	Eric W Gray	Vincent Lefevre
Demetrio Bucaneg Jr.	Randall Groves	Peter Petrov
Juan Carreon	Joseph Gwinn	Walter Struppler
Donald Cragun	Werner Hoelzl	Gerald Stueve
Sourav Dutta	Noriyuki Ikeuchi	Mark-Rene Uchida
Andrew Fieldsend	Andrew Josey	Karl Weber
David Fuschi	Piotr Karocki	Oren Yuen

When the IEEE-SA Standards Board approved this standard on 6 December 2017, it had the following membership:

Jean-Philippe Faure, *Chair*
Gary Hoffman, *Vice Chair*
John D. Kulick, *Past Chair*
Konstantinos Karachalios, *Secretary*

Chuck Adams	Thomas Koshy	Dorothy Stanley
Masayuki Ariyoshi	Joseph L. Koepfinger*	Adrian Stephens
Ted Burse	Kevin Lu	Mehmet Ulema
Stephen Dukes	Daleep Mohla	Phil Wennblom
Doug Edwards	Damir Novosel	Howard Wolfman
J. Travis Griffith	Ronald C. Petersen	Yu Yuan
Michael Janezic	Annette D. Reilly	
	Robby Robson	

* Member Emeritus

Introduction

This introduction is not part of IEEE Std 1003.1-2017, Standard for Information Technology—Portable Operating System Interface (POSIX®).

This standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.^a

The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group is to develop and maintain the core open systems interfaces that are the POSIX® 1003.1 (and former 1003.2) standards, ISO/IEC 9945, and the core of the Single UNIX Specification.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group Standard designation, and an ISO/IEC designation.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see www.opengroup.org/austin.

Background

The developers of POSIX.1-2017 represent a cross-section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, POSIX.1-2017 describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application developers to write portable applications – it was developed with that goal in mind – it has been designated POSIX^b, an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of standards.

^a The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.

^b The Name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

Audience

The intended audience for POSIX.1-2017 is all persons concerned with an industry-wide standard operating system based on the UNIX system. This includes at least four groups of people:

- Persons buying hardware and software systems
- Persons managing companies that are deciding on future corporate computing directions
- Persons implementing operating systems, and especially
- Persons developing applications where portability is an objective

Purpose

Several principles guided the development of POSIX.1-2017:

- **Application-Oriented** – The basic goal was to promote portability of application programs across UNIX system environments by developing a clear, consistent, and unambiguous standard for the interface specification of a portable operating system based on the UNIX system documentation. POSIX.1-2017 codifies the common, existing definition of the UNIX system.
- **Interface, Not Implementation** – POSIX.1-2017 defines an interface, not an implementation. No distinction is made between library functions and system calls; both are referred to as functions. No details of the implementation of any function are given (although historical practice is sometimes indicated in the RATIONALE section). Symbolic names are given for constants (such as signals and error numbers) rather than numbers.
- **Source, Not Object, Portability** – POSIX.1-2017 has been written so that a program written and translated for execution on one conforming implementation may also be translated for execution on another conforming implementation. POSIX.1-2017 does not guarantee that executable (object or binary) code will execute under a different conforming implementation than that for which it was translated, even if the underlying hardware is identical.
- **The C Language** – The system interfaces and header definitions are written in terms of the standard C language as specified in the ISO C standard.
- **No Superuser, No System Administration** – There was no intention to specify all aspects of an operating system. System administration facilities and functions are excluded from this standard, and functions usable only by the superuser have not been included. Still, an implementation of the standard interface may also implement features not in POSIX.1-2017. POSIX.1-2017 is also not concerned with hardware constraints or system maintenance.
- **Minimal Interface, Minimally Defined** – In keeping with the historical design principles of the UNIX system, the mandatory core facilities of POSIX.1-2017 have been kept as minimal as possible. Additional capabilities have been added as optional extensions.
- **Broadly Implementable** – The developers of POSIX.1-2017 endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:
 - All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
 - Compatible systems that are not derived from the original UNIX system code
 - Emulations hosted on entirely different operating systems
 - Networked systems
 - Distributed systems
 - Systems running on a broad range of hardware

No direct references to this goal appear in POSIX.1-2017, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations – When the original version – IEEE Std 1003.1-1988 – was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and IEEE Std 1003.1-2001 and its technical corrigenda have consolidated this consensus, and this version reflects the significantly increased level of consensus arrived at since the original versions. The authors of the original versions tried, as much as possible, to follow the principles below when creating new specifications:

- By standardizing an interface like one in an historical implementation; for example, directories
- By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
- By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

POSIX.1-2017 is specifically not a codification of a particular vendor’s product.

It should be noted that implementations will have different kinds of extensions. Some will reflect “historical usage” and will be preserved for execution of pre-existing applications. These functions should be considered “obsolescent” and the standard functions used for new applications. Some extensions will represent functions beyond the scope of POSIX.1-2017. These need to be used with careful management to be able to adapt to future extensions of POSIX.1-2017 and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code – A goal of POSIX.1-2017 was to minimize additional work for application developers. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

POSIX.1-2017

POSIX.1-2017 defines the Portable Operating System Interface (POSIX) requirements and consists of the following topics arranged as a series of volumes within the standard:

- Base Definitions
- System Interfaces
- Shell and Utilities
- Rationale (Informative)

Base Definitions

The Base Definitions volume provides common definitions for this standard, therefore readers should be familiar with it before using the other volumes.

This volume is structured as follows:

- Chapter 1 is an introduction.
- Chapter 2 defines the conformance requirements.
- Chapter 3 defines general terms used.
- Chapter 4 describes general concepts used.
- Chapter 5 describes the notation used to specify file input and output formats in this volume and the Shell and Utilities volume.
- Chapter 6 describes the portable character set and the process of character set definition.

- Chapter 7 describes the syntax for defining internationalization locales as well as the POSIX locale provided on all systems.
- Chapter 8 describes the use of environment variables for internationalization and other purposes.
- Chapter 9 describes the syntax of pattern matching using regular expressions employed by many utilities and matched by the *regcomp()* and *regexexec()* functions.
- Chapter 10 describes files and devices found on all systems.
- Chapter 11 describes the asynchronous terminal interface for many of the functions in the System Interfaces volume and the *stty* utility in the Shell and Utilities volume.
- Chapter 12 describes the policies for command line argument construction and parsing.
- Chapter 13 defines the contents of headers which declare the functions and global variables, and define types, constants, macros, and data structures that are needed by programs using the services provided by the System Interfaces volume.

Comprehensive references are available in the index.

System Interfaces

The System Interfaces volume describes the interfaces offered to application programs by POSIX-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards.
- Chapter 2 contains important concepts, terms, and caveats relating to the rest of this volume.
- Chapter 3 defines the functional interfaces to the POSIX-conformant system.

Comprehensive references are available in the index.

Shell and Utilities

The Shell and Utilities volume describes the commands and utilities offered to application programs on POSIX-conformant systems. Readers are expected to be familiar with the Base Definitions volume.

This volume is structured as follows:

- Chapter 1 explains the status of this volume and its relationship to other formal standards. It also describes the defaults used by the utility descriptions.
- Chapter 2 describes the command language used in POSIX-conformant systems, and special built-in utilities.
- Chapter 3 describes a set of services and utilities that are implemented on systems supporting the Batch Environment Services and Utilities option.
- Chapter 4 consists of reference pages for all utilities, other than the special built-in utilities described in Chapter 2, available on POSIX-conformant systems.

Comprehensive references are available in the index.

Rationale (Informative)

The Rationale volume is published to assist in the process of review. It contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of POSIX.1-2017 that may not be immediately apparent.

This volume is organized in parallel to the normative volumes of this standard, with a separate part for each of the three normative volumes.

Within this volume, the following terms are used:

- Base standard – The portions of POSIX.1-2017 that are not optional, equivalent to the definitions of classic POSIX.1 and POSIX.2.
- POSIX.0 – Although this term is not used in the normative text of POSIX.1-2017, it is used in this volume to refer to IEEE Std 1003.0™-1995.
- POSIX.1b – Although this term is not used in the normative text of POSIX.1-2017, it is used in this volume to refer to the elements of the POSIX Realtime Extension amendment. (This was earlier referred to as POSIX.4 during the standard development process.)
- POSIX.1c – Although this term is not used in the normative text of POSIX.1-2017, it is used in this volume to refer to the POSIX Threads Extension amendment. (This was earlier referred to as POSIX.4a during the standard development process.)
- Standard developers – The individuals and companies in the development organizations responsible for POSIX.1-2017: the IEEE P1003.1 working groups, The Open Group Base working group, advised by the hundreds of individual technical experts who balloted the draft standards within the Austin Group, and the member bodies and technical experts of ISO/IEC JTC 1/SC 22.
- XSI option – The portions of POSIX.1-2017 addressing the extension added for support of the Single UNIX Specification.

Typographical Conventions

The following typographical conventions are used throughout this standard. In the text, this standard is referred to as POSIX.1-2017, which is technically identical to The Open Group Base Specifications, Issue 7.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in POSIX.1-2017.

Reference	Example	Notes
C-Language Data Structure	aiocb	
C-Language Data Structure Member	<i>aiio_lio_opcode</i>	
C-Language Data Type	long	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	return	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	NET_ADDRSTRLEN	
C-Language Preprocessing Directive	#define	
Commands within a Utility	a, c	
Conversion Specifier, Specifier/Modifier Character	%A, g, E	1

Reference	Example	Notes
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	Hello, World	
Filename	/tmp	
Literal Character	'c', '\r'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[]	
Parameter	<directory pathname>	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	
User Input and Example Code	echo Hello, World	5
Utility Name	<i>awk</i>	
Utility Operand	<i>file_name</i>	
Utility Option	-c	
Utility Option with Option-Argument	-w width	

Note that:

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf()* and *fscanf()* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. The literal characters <apostrophe> (also known as single-quote) and <backslash> are either shown as the C constants '\ ' and '\\ ', respectively, or as the special characters <apostrophe>, single-quote, and <backslash> depending on context.
3. The style selected for some of the special characters, such as <newline>, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
5. Brackets shown in this font, "[]", are part of the syntax and do not indicate optional items. In syntax the '| ' symbol is used to separate alternatives, and ellipses ("...") are used to show that additional arguments are optional.

Shading is used to identify extensions and options.

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

Ranges of values are indicated with parentheses or brackets as follows:

1. (a,b) means the range of all values from a to b , including neither a nor b .
2. $[a,b]$ means the range of all values from a to b , including a and b .
3. $[a,b)$ means the range of all values from a to b , including a , but not b .
4. $(a,b]$ means the range of all values from a to b , including b , but not a .

Note: A symbolic limit beginning with POSIX is treated differently, depending on context. In a C-language header, the symbol `POSIXstring` (where *string* may contain underscores) is represented by the C identifier `_POSIXstring`, with a leading underscore required to prevent ISO C standard name space pollution. However, in other contexts, such as languages other than C, the leading underscore is not used because this requirement does not exist.

Contents

Volume	1	Base Definitions, Issue 7.....	1
Chapter	1	Introduction.....	3
	1.1	Scope.....	3
	1.2	Conformance.....	4
	1.3	Normative References.....	4
	1.4	Change History.....	5
	1.5	Terminology.....	5
	1.6	Definitions and Concepts.....	6
	1.7	Portability.....	6
	1.7.1	Codes.....	7
	1.7.2	Margin Code Notation.....	13
Chapter	2	Conformance.....	15
	2.1	Implementation Conformance.....	15
	2.1.1	Requirements.....	15
	2.1.2	Documentation.....	16
	2.1.3	POSIX Conformance.....	17
	2.1.4	XSI Conformance.....	19
	2.1.5	Option Groups.....	20
	2.1.6	Options.....	26
	2.2	Application Conformance.....	29
	2.2.1	Strictly Conforming POSIX Application.....	29
	2.2.2	Conforming POSIX Application.....	30
	2.2.3	Conforming POSIX Application Using Extensions.....	30
	2.2.4	Strictly Conforming XSI Application.....	30
	2.2.5	Conforming XSI Application Using Extensions.....	31
	2.3	Language-Dependent Services for the C Programming Language.....	31
	2.4	Other Language-Related Specifications.....	31
Chapter	3	Definitions.....	33
	3.1	Abortive Release.....	33
	3.2	Absolute Pathname.....	33
	3.3	Access Mode.....	33
	3.4	Additional File Access Control Mechanism.....	33
	3.5	Address Space.....	33
	3.6	Advisory Information.....	34
	3.7	Affirmative Response.....	34
	3.8	Alert.....	34
	3.9	Alert Character (<alert>).....	34
	3.10	Alias Name.....	34
	3.11	Alignment.....	34
	3.12	Alternate File Access Control Mechanism.....	35

3.13	Alternate Signal Stack.....	35
3.14	Ancillary Data.....	35
3.15	Angle Brackets.....	35
3.16	Apostrophe Character (<apostrophe>).....	35
3.17	Application.....	36
3.18	Application Address.....	36
3.19	Application Program Interface (API).....	36
3.20	Appropriate Privileges.....	36
3.21	Argument.....	36
3.22	Arm (a Timer).....	36
3.23	Asterisk Character (<asterisk>).....	37
3.24	Async-Cancel-Safe Function.....	37
3.25	Asynchronous Events.....	37
3.26	Asynchronous Input and Output.....	37
3.27	Async-Signal-Safe Function.....	37
3.28	Asynchronously-Generated Signal.....	37
3.29	Asynchronous I/O Completion.....	37
3.30	Asynchronous I/O Operation.....	38
3.31	Authentication.....	38
3.32	Authorization.....	38
3.33	Background Job.....	38
3.34	Background Process.....	38
3.35	Background Process Group (or Background Job).....	38
3.36	Backquote Character.....	38
3.37	Backslash Character (<backslash>).....	38
3.38	Backspace Character (<backspace>).....	39
3.39	Barrier.....	39
3.40	Basename.....	39
3.41	Basic Regular Expression (BRE).....	39
3.42	Batch Access List.....	39
3.43	Batch Administrator.....	39
3.44	Batch Client.....	40
3.45	Batch Destination.....	40
3.46	Batch Destination Identifier.....	40
3.47	Batch Directive.....	40
3.48	Batch Job.....	40
3.49	Batch Job Attribute.....	41
3.50	Batch Job Identifier.....	41
3.51	Batch Job Name.....	41
3.52	Batch Job Owner.....	41
3.53	Batch Job Priority.....	41
3.54	Batch Job State.....	41
3.55	Batch Name Service.....	41
3.56	Batch Name Space.....	42
3.57	Batch Node.....	42
3.58	Batch Operator.....	42
3.59	Batch Queue.....	42
3.60	Batch Queue Attribute.....	42
3.61	Batch Queue Position.....	42
3.62	Batch Queue Priority.....	43
3.63	Batch Rerunability.....	43
3.64	Batch Restart.....	43

3.65	Batch Server	43
3.66	Batch Server Name.....	43
3.67	Batch Service	43
3.68	Batch Service Request.....	44
3.69	Batch Submission	44
3.70	Batch System.....	44
3.71	Batch Target User	44
3.72	Batch User	44
3.73	Bind	44
3.74	Blank Character (<blank>).....	44
3.75	Blank Line.....	45
3.76	Blocked Process (or Thread)	45
3.77	Blocking	45
3.78	Block-Mode Terminal	45
3.79	Block Special File.....	45
3.80	Braces	45
3.81	Brackets.....	45
3.82	Broadcast	46
3.83	Built-In Utility (or Built-In).....	46
3.84	Byte.....	46
3.85	Byte Input/Output Functions	46
3.86	Carriage-Return Character (<carriage-return>)	46
3.87	Character	47
3.88	Character Array.....	47
3.89	Character Class.....	47
3.90	Character Set.....	47
3.91	Character Special File	47
3.92	Character String.....	47
3.93	Child Process	48
3.94	Circum ex Character (<circum ex>).....	48
3.95	Clock	48
3.96	Clock Jump.....	48
3.97	Clock Tick.....	48
3.98	Coded Character Set	48
3.99	Codeset	49
3.100	Collating Element.....	49
3.101	Collation	49
3.102	Collation Sequence.....	49
3.103	Column Position.....	50
3.104	Command.....	50
3.105	Command Language Interpreter	50
3.106	Composite Graphic Symbol.....	50
3.107	Condition Variable	50
3.108	Connected Socket	51
3.109	Connection	51
3.110	Connection Mode.....	51
3.111	Connectionless Mode	51
3.112	Control Character.....	51
3.113	Control Operator	51
3.114	Controlling Process.....	51
3.115	Controlling Terminal	52
3.116	Conversion Descriptor	52

3.117	Core File.....	52
3.118	CPU Time (Execution Time)	52
3.119	CPU-Time Clock.....	52
3.120	CPU-Time Timer.....	52
3.121	Current Job.....	52
3.122	Current Working Directory.....	53
3.123	Cursor Position.....	53
3.124	Datagram.....	53
3.125	Data Segment.....	53
3.126	Deferred Batch Service	53
3.127	Device	53
3.128	Device ID.....	53
3.129	Directory.....	53
3.130	Directory Entry (or Link)	53
3.131	Directory Stream	54
3.132	Disarm (a Timer)	54
3.133	Display.....	54
3.134	Display Line.....	54
3.135	Dollar-Sign Character (<dollar-sign>)	54
3.136	Dot.....	54
3.137	Dot-Dot.....	55
3.138	Double-Quote Character.....	55
3.139	Downshifting.....	55
3.140	Driver.....	55
3.141	Effective Group ID.....	55
3.142	Effective User ID.....	55
3.143	Eight-Bit Transparency.....	55
3.144	Empty Directory.....	56
3.145	Empty Line.....	56
3.146	Empty String (or Null String).....	56
3.147	Empty Wide-Character String.....	56
3.148	Encoding Rule	56
3.149	Entire Regular Expression.....	56
3.150	Epoch	57
3.151	Equivalence Class.....	57
3.152	Era.....	57
3.153	Event Management.....	57
3.154	Executable File.....	57
3.155	Execute.....	58
3.156	Execution Time.....	58
3.157	Execution Time Monitoring.....	58
3.158	Expand.....	58
3.159	Extended Regular Expression (ERE)	58
3.160	Extended Security Controls.....	58
3.161	Feature Test Macro.....	59
3.162	Field.....	59
3.163	FIFO Special File (or FIFO).....	59
3.164	File.....	59
3.165	File Description	59
3.166	File Descriptor	60
3.167	File Group Class.....	60
3.168	File Mode.....	60

3.169	File Mode Bits	60
3.170	Filename	60
3.171	Filename String.....	61
3.172	File Offset	61
3.173	File Other Class	61
3.174	File Owner Class	61
3.175	File Permission Bits.....	61
3.176	File Serial Number	61
3.177	File System	61
3.178	File Type	62
3.179	Filter	62
3.180	First Open (of a File)	62
3.181	Flow Control	62
3.182	Foreground Job.....	62
3.183	Foreground Process	62
3.184	Foreground Process Group (or Foreground Job).....	62
3.185	Foreground Process Group ID	62
3.186	Form-Feed Character (<form-feed>).....	63
3.187	Graphic Character	63
3.188	Group Database.....	63
3.189	Group ID.....	63
3.190	Group Name	63
3.191	Hard Limit.....	64
3.192	Hard Link	64
3.193	Home Directory	64
3.194	Host Byte Order.....	64
3.195	Incomplete Line.....	64
3.196	Inf.....	64
3.197	Instrumented Application	64
3.198	Interactive Shell	65
3.199	Internationalization	65
3.200	Interprocess Communication	65
3.201	Invoke	65
3.202	Job.....	65
3.203	Job Control	65
3.204	Job Control Job ID	66
3.205	Last Close (of a File).....	66
3.206	Line.....	66
3.207	Linger.....	66
3.208	Link	66
3.209	Link Count	66
3.210	Live Process.....	66
3.211	Local Customs	67
3.212	Local Interprocess Communication (Local IPC).....	67
3.213	Locale	67
3.214	Localization.....	67
3.215	Login	67
3.216	Login Name	67
3.217	Map	67
3.218	Marked Message	68
3.219	Matched	68
3.220	Memory Mapped Files	68

3.221	Memory Object	68
3.222	Memory-Resident.....	68
3.223	Message	69
3.224	Message Catalog.....	69
3.225	Message Catalog Descriptor	69
3.226	Message Queue.....	69
3.227	Mode	69
3.228	Monotonic Clock	69
3.229	Mount Point	70
3.230	Multi-Character Collating Element	70
3.231	Multi-Threaded Library	70
3.232	Multi-Threaded Process	70
3.233	Multi-Threaded Program	70
3.234	Mutex	70
3.235	Name.....	71
3.236	Named STREAM.....	71
3.237	NaN (Not a Number)	71
3.238	Native Language	71
3.239	Negative Response.....	71
3.240	Network.....	71
3.241	Network Address	71
3.242	Network Byte Order	72
3.243	Newline Character (<newline>)	72
3.244	Nice Value	72
3.245	Non-Blocking.....	72
3.246	Non-Spacing Characters	72
3.247	NUL.....	73
3.248	Null Byte.....	73
3.249	Null Pointer.....	73
3.250	Null String.....	73
3.251	Null Wide-Character Code	73
3.252	Number-Sign Character (<number-sign>)	73
3.253	Object File.....	73
3.254	Octet	73
3.255	Offset Maximum	74
3.256	Opaque Address.....	74
3.257	Open File	74
3.258	Open File Description.....	74
3.259	Operand.....	74
3.260	Operator	74
3.261	Option	75
3.262	Option-Argument	75
3.263	Orientation	75
3.264	Orphaned Process Group.....	75
3.265	Page	75
3.266	Page Size.....	75
3.267	Parameter	76
3.268	Parent Directory	76
3.269	Parent Process.....	76
3.270	Parent Process ID	76
3.271	Pathname.....	76
3.272	Pathname Component.....	77

3.273	Path Pre x	77
3.274	Pattern.....	77
3.275	Period Character (<period>)	77
3.276	Permissions	78
3.277	Persistence.....	78
3.278	Pipe.....	78
3.279	Polling.....	78
3.280	Portable Character Set	78
3.281	Portable Filename.....	79
3.282	Portable Filename Character Set.....	79
3.283	Positional Parameter.....	79
3.284	Preallocation	79
3.285	Preempted Process (or Thread).....	79
3.286	Previous Job	79
3.287	Printable Character	80
3.288	Printable File	80
3.289	Priority	80
3.290	Priority Band.....	80
3.291	Priority Inversion	80
3.292	Priority Scheduling.....	80
3.293	Priority-Based Scheduling	80
3.294	Privilege.....	81
3.295	Process	81
3.296	Process Group.....	81
3.297	Process Group ID	81
3.298	Process Group Leader.....	81
3.299	Process Group Lifetime	81
3.300	Process ID.....	82
3.301	Process Lifetime.....	82
3.302	Process Memory Locking.....	82
3.303	Process Termination.....	82
3.304	Process-To-Process Communication	82
3.305	Process Virtual Time	83
3.306	Program	83
3.307	Protocol.....	83
3.308	Pseudo-Terminal	83
3.309	Radix Character	83
3.310	Read-Only File System	83
3.311	Read-Write Lock.....	83
3.312	Real Group ID.....	84
3.313	Real Time	84
3.314	Realtime Signal Extension	84
3.315	Real User ID	84
3.316	Record	84
3.317	Redirection	84
3.318	Redirection Operator	85
3.319	Referenced Shared Memory Object	85
3.320	Refresh	85
3.321	Regular Expression	85
3.322	Region	85
3.323	Regular File	85
3.324	Relative Pathname	86

3.325	Relocatable File.....	86
3.326	Relocation.....	86
3.327	Requested Batch Service	86
3.328	(Time) Resolution.....	86
3.329	Robust Mutex.....	86
3.330	Root Directory	86
3.331	Runnable Process (or Thread).....	86
3.332	Running Process (or Thread).....	87
3.333	Saved Resource Limits	87
3.334	Saved Set-Group-ID.....	87
3.335	Saved Set-User-ID	87
3.336	Scheduling.....	87
3.337	Scheduling Allocation Domain.....	87
3.338	Scheduling Contention Scope	87
3.339	Scheduling Policy.....	88
3.340	Screen.....	88
3.341	Scroll.....	88
3.342	Semaphore.....	88
3.343	Session	89
3.344	Session Leader	89
3.345	Session Lifetime.....	89
3.346	Shared Memory Object.....	89
3.347	Shell.....	89
3.348	Shell, the	89
3.349	Shell Script.....	90
3.350	Signal.....	90
3.351	Signal Stack	90
3.352	Single-Quote Character	90
3.353	Single-Threaded Process	90
3.354	Single-Threaded Program.....	90
3.355	Slash Character (<slash>).....	91
3.356	Socket	91
3.357	Socket Address	91
3.358	Soft Limit	91
3.359	Source Code	91
3.360	Space Character (<space>).....	92
3.361	Spawn	92
3.362	Special Built-In.....	92
3.363	Special Parameter.....	92
3.364	Spin Lock.....	92
3.365	Sporadic Server.....	92
3.366	Standard Error	92
3.367	Standard Input.....	92
3.368	Standard Output	93
3.369	Standard Utilities	93
3.370	Stream	93
3.371	STREAM.....	93
3.372	STREAM End.....	93
3.373	STREAM Head	93
3.374	STREAMS Multiplexor.....	94
3.375	String.....	94
3.376	Subshell.....	94

3.377	Successfully Transferred	94
3.378	Supplementary Group ID	94
3.379	Suspended Job	94
3.380	Symbolic Constant	95
3.381	Symbolic Link	95
3.382	Synchronized Input and Output.....	95
3.383	Synchronized I/O Completion	95
3.384	Synchronized I/O Data Integrity Completion.....	95
3.385	Synchronized I/O File Integrity Completion	96
3.386	Synchronized I/O Operation	96
3.387	Synchronous I/O Operation.....	96
3.388	Synchronously-Generated Signal	96
3.389	System.....	96
3.390	System Boot.....	96
3.391	System Clock.....	96
3.392	System Console	97
3.393	System Crash	97
3.394	System Databases.....	97
3.395	System Documentation	97
3.396	System Process.....	97
3.397	System Reboot	97
3.398	System Trace Event	97
3.399	System-Wide	98
3.400	Tab Character (<tab>).....	98
3.401	Terminal (or Terminal Device)	98
3.402	Text Column.....	98
3.403	Text File.....	98
3.404	Thread	99
3.405	Thread ID	99
3.406	Thread List	99
3.407	Thread-Safe	99
3.408	Thread-Specific Data Key.....	99
3.409	Tilde Character (<tilde>).....	99
3.410	Timeouts	100
3.411	Timer	100
3.412	Timer Overrun.....	100
3.413	Token.....	100
3.414	Trace Analyzer Process.....	100
3.415	Trace Controller Process.....	100
3.416	Trace Event.....	100
3.417	Trace Event Type	100
3.418	Trace Event Type Mapping.....	101
3.419	Trace Filter	101
3.420	Trace Generation Version	101
3.421	Trace Log	101
3.422	Trace Point.....	101
3.423	Trace Stream.....	101
3.424	Trace Stream Identifier	101
3.425	Trace System	101
3.426	Traced Process.....	102
3.427	Tracing Status of a Trace Stream	102
3.428	Typed Memory Name Space	102

3.429	Typed Memory Object	102
3.430	Typed Memory Pool	102
3.431	Typed Memory Port	102
3.432	Unbind	102
3.433	Unit Data	102
3.434	Upshifting	103
3.435	User Database	103
3.436	User ID	103
3.437	User Name	103
3.438	User Trace Event	103
3.439	Utility	104
3.440	Variable	104
3.441	Vertical-Tab Character (<vertical-tab>)	104
3.442	White Space	104
3.443	Wide-Character Code (C Language)	104
3.444	Wide-Character Input/Output Functions	105
3.445	Wide-Character String	105
3.446	Word	105
3.447	Working Directory (or Current Working Directory)	105
3.448	Worldwide Portability Interface	105
3.449	Write	105
3.450	XSI	105
3.451	XSI-Conformant	106
3.452	Zombie Process	106
3.453	±0	106
Chapter 4	General Concepts	107
4.1	Concurrent Execution	107
4.2	Default Initialization	107
4.3	Directory Protection	108
4.4	Extended Security Controls	108
4.5	File Access Permissions	108
4.6	File Hierarchy	109
4.7	Filenames	109
4.8	Filename Portability	109
4.9	File Times Update	109
4.10	Host and Network Byte Orders	110
4.11	Measurement of Execution Time	110
4.12	Memory Synchronization	111
4.13	Pathname Resolution	111
4.14	Process ID Reuse	113
4.15	Scheduling Policy	113
4.16	Seconds Since the Epoch	113
4.17	Semaphore	114
4.18	Thread-Safety	114
4.19	Tracing	115
4.20	Treatment of Error Conditions for Mathematical Functions	117
4.20.1	Domain Error	117
4.20.2	Pole Error	117
4.20.3	Range Error	118
4.21	Treatment of NaN Arguments for the Mathematical	

		Functions	118
	4.22	Utility	119
	4.23	Variable Assignment.....	119
Chapter	5	File Format Notation.....	121
Chapter	6	Character Set	125
	6.1	Portable Character Set	125
	6.2	Character Encoding	128
	6.3	C Language Wide-Character Codes	128
	6.4	Character Set Description File.....	129
	6.4.1	State-Dependent Character Encodings	133
Chapter	7	Locale	135
	7.1	General.....	135
	7.2	POSIX Locale	136
	7.3	Locale De nition	136
	7.3.1	LC_CTYPE	139
	7.3.2	LC_COLLATE.....	147
	7.3.3	LC_MONETARY	155
	7.3.4	LC_NUMERIC.....	158
	7.3.5	LC_TIME	159
	7.3.6	LC_MESSAGES	165
	7.4	Locale De nition Grammar.....	166
	7.4.1	Locale Lexical Conventions	166
	7.4.2	Locale Grammar.....	167
Chapter	8	Environment Variables.....	173
	8.1	Environment Variable Definition.....	173
	8.2	Internationalization Variables	174
	8.3	Other Environment Variables.....	177
Chapter	9	Regular Expressions.....	181
	9.1	Regular Expression Definitions.....	181
	9.2	Regular Expression General Requirements.....	182
	9.3	Basic Regular Expressions	183
	9.3.1	BREs Matching a Single Character or Collating Element	183
	9.3.2	BRE Ordinary Characters.....	183
	9.3.3	BRE Special Characters	183
	9.3.4	Periods in BREs	184
	9.3.5	RE Bracket Expression.....	184
	9.3.6	BREs Matching Multiple Characters	186
	9.3.7	BRE Precedence	187
	9.3.8	BRE Expression Anchoring.....	188
	9.4	Extended Regular Expressions.....	188
	9.4.1	EREs Matching a Single Character or Collating Element	188
	9.4.2	ERE Ordinary Characters.....	188
	9.4.3	ERE Special Characters	189
	9.4.4	Periods in EREs	189
	9.4.5	ERE Bracket Expression	189

	9.4.6	EREs Matching Multiple Characters	190
	9.4.7	ERE Alternation.....	191
	9.4.8	ERE Precedence	191
	9.4.9	ERE Expression Anchoring.....	191
	9.5	Regular Expression Grammar	192
	9.5.1	BRE/ERE Grammar Lexical Conventions.....	192
	9.5.2	RE and Bracket Expression Grammar.....	193
	9.5.3	ERE Grammar.....	195
Chapter	10	Directory Structure and Devices	197
	10.1	Directory Structure and Files.....	197
	10.2	Output Devices and Terminal Types.....	198
Chapter	11	General Terminal Interface	199
	11.1	Interface Characteristics	199
	11.1.1	Opening a Terminal Device File.....	199
	11.1.2	Process Groups	200
	11.1.3	The Controlling Terminal.....	200
	11.1.4	Terminal Access Control	201
	11.1.5	Input Processing and Reading Data	201
	11.1.6	Canonical Mode Input Processing.....	202
	11.1.7	Non-Canonical Mode Input Processing.....	202
	11.1.8	Writing Data and Output Processing.....	203
	11.1.9	Special Characters	204
	11.1.10	Modem Disconnect	205
	11.1.11	Closing a Terminal Device File.....	205
	11.2	Parameters that Can be Set	205
	11.2.1	The termios Structure	205
	11.2.2	Input Modes.....	206
	11.2.3	Output Modes.....	207
	11.2.4	Control Modes	209
	11.2.5	Local Modes.....	210
	11.2.6	Special Control Characters.....	212
Chapter	12	Utility Conventions.....	213
	12.1	Utility Argument Syntax.....	213
	12.2	Utility Syntax Guidelines.....	216
Chapter	13	Headers	219
Volume	2	System Interfaces, Issue 7.....	467
Chapter	1	Introduction.....	469
	1.1	Relationship to Other Formal Standards.....	469
	1.2	Format of Entries.....	469
Chapter	2	General Information	471
	2.1	Use and Implementation of Interfaces	471
	2.1.1	Use and Implementation of Functions.....	471
	2.1.2	Use and Implementation of Macros	472
	2.2	The Compilation Environment	472

2.2.1	POSIX.1 Symbols.....	472
2.2.2	The Name Space.....	473
2.3	Error Numbers.....	481
2.3.1	Additional Error Numbers	488
2.4	Signal Concepts	488
2.4.1	Signal Generation and Delivery.....	488
2.4.2	Realtime Signal Generation and Delivery	489
2.4.3	Signal Actions.....	490
2.4.4	Signal Effects on Other Functions.....	495
2.5	Standard I/O Streams	495
2.5.1	Interaction of File Descriptors and Standard I/O Streams	497
2.5.2	Stream Orientation and Encoding Rules	498
2.6	STREAMS.....	500
2.6.1	Accessing STREAMS	501
2.7	XSI Interprocess Communication	501
2.7.1	IPC General Description	502
2.8	Realtime.....	503
2.8.1	Realtime Signals	503
2.8.2	Asynchronous I/O.....	503
2.8.3	Memory Management.....	505
2.8.4	Process Scheduling.....	506
2.8.5	Clocks and Timers.....	511
2.9	Threads	512
2.9.1	Thread-Safety.....	513
2.9.2	Thread IDs.....	513
2.9.3	Thread Mutexes.....	514
2.9.4	Thread Scheduling	515
2.9.5	Thread Cancellation.....	517
2.9.6	Thread Read-Write Locks.....	521
2.9.7	Thread Interactions with Regular File Operations.....	522
2.9.8	Use of Application-Managed Thread Stacks.....	522
2.9.9	Synchronization Object Copies and Alternative Mappings.....	523
2.10	Sockets	523
2.10.1	Address Families	523
2.10.2	Addressing	524
2.10.3	Protocols	524
2.10.4	Routing	524
2.10.5	Interfaces	524
2.10.6	Socket Types.....	524
2.10.7	Socket I/O Mode.....	525
2.10.8	Socket Owner.....	526
2.10.9	Socket Queue Limits	526
2.10.10	Pending Error	526
2.10.11	Socket Receive Queue.....	526
2.10.12	Socket Out-of-Band Data State.....	527
2.10.13	Connection Indication Queue	527
2.10.14	Signals.....	527
2.10.15	Asynchronous Errors.....	527
2.10.16	Use of Options	528
2.10.17	Use of Sockets for Local UNIX Connections	531

2.10.18	Use of Sockets over Internet Protocols.....	532
2.10.19	Use of Sockets over Internet Protocols Based on IPv4.....	532
2.10.20	Use of Sockets over Internet Protocols Based on IPv6.....	532
2.11	Tracing	536
2.11.1	Tracing Data Definitions.....	538
2.11.2	Trace Event Type Definitions.....	542
2.11.3	Trace Functions.....	546
2.12	Data Types.....	547
2.12.1	DefinedTypes	547
2.12.2	The char Type.....	548
2.13	Status Information	548
2.14	File Descriptor Allocation	549
Chapter 3	System Interfaces.....	551
Volume 3	Shell and Utilities, Issue 7	2325
Chapter 1	Introduction.....	2327
1.1	Relationship to Other Documents	2327
1.1.1	System Interfaces.....	2327
1.1.2	Concepts Derived from the ISO C Standard	2331
1.2	Utility Limits.....	2333
1.3	Grammar Conventions.....	2335
1.4	Utility Description Defaults.....	2336
1.5	Considerations for Utilities in Support of Files of Arbitrary Size.....	2343
1.6	Built-In Utilities	2344
Chapter 2	Shell Command Language	2345
2.1	Shell Introduction.....	2345
2.2	Quoting.....	2346
2.2.1	Escape Character (Backslash).....	2346
2.2.2	Single-Quotes.....	2346
2.2.3	Double-Quotes.....	2346
2.3	Token Recognition.....	2347
2.3.1	Alias Substitution.....	2348
2.4	Reserved Words.....	2349
2.5	Parameters and Variables.....	2349
2.5.1	Positional Parameters	2349
2.5.2	Special Parameters	2350
2.5.3	Shell Variables.....	2351
2.6	Word Expansions	2353
2.6.1	Tilde Expansion	2354
2.6.2	Parameter Expansion.....	2354
2.6.3	Command Substitution	2357
2.6.4	Arithmetic Expansion.....	2358
2.6.5	Field Splitting	2359
2.6.6	Pathname Expansion	2360
2.6.7	Quote Removal.....	2360

2.7	Redirection	2360
2.7.1	Redirecting Input	2361
2.7.2	Redirecting Output	2361
2.7.3	Appending Redirected Output	2361
2.7.4	Here-Document	2362
2.7.5	Duplicating an Input File Descriptor	2363
2.7.6	Duplicating an Output File Descriptor	2363
2.7.7	Open File Descriptors for Reading and Writing	2363
2.8	Exit Status and Errors	2363
2.8.1	Consequences of Shell Errors	2363
2.8.2	Exit Status for Commands	2364
2.9	Shell Commands	2365
2.9.1	Simple Commands	2365
2.9.2	Pipelines	2368
2.9.3	Lists	2369
2.9.4	Compound Commands	2371
2.9.5	Function Definition Command	2374
2.10	Shell Grammar	2375
2.10.1	Shell Grammar Lexical Conventions	2375
2.10.2	Shell Grammar Rules	2375
2.11	Signals and Error Handling	2381
2.12	Shell Execution Environment	2381
2.13	Pattern Matching Notation	2382
2.13.1	Patterns Matching a Single Character	2382
2.13.2	Patterns Matching Multiple Characters	2383
2.13.3	Patterns Used for Filename Expansion	2383
2.14	Special Built-In Utilities	2384
Chapter 3	Batch Environment Services	2427
3.1	General Concepts	2427
3.1.1	Batch Client-Server Interaction	2427
3.1.2	Batch Queues	2428
3.1.3	Batch Job Creation	2428
3.1.4	Batch Job Tracking	2428
3.1.5	Batch Job Routing	2429
3.1.6	Batch Job Execution	2429
3.1.7	Batch Job Exit	2430
3.1.8	Batch Job Abort	2430
3.1.9	Batch Authorization	2430
3.1.10	Batch Administration	2430
3.1.11	Batch Notification	2431
3.2	Batch Services	2431
3.2.1	Batch Job States	2432
3.2.2	Deferred Batch Services	2433
3.2.3	Requested Batch Services	2442
3.3	Common Behavior for Batch Environment Utilities	2449
3.3.1	Batch Job Identifier	2449
3.3.2	Destination	2450
3.3.3	Multiple Keyword-Value Pairs	2451
Chapter 4	Utilities	2453

Volume	4	Rationale (Informative), Issue 7	3473
Part	A	Base Definitions	3475
Appendix	A	Rationale for Base Definitions	3477
	A.1	Introduction	3477
	A.1.1	Scope	3477
	A.1.2	Conformance.....	3480
	A.1.3	Normative References	3480
	A.1.4	Change History	3480
	A.1.5	Terminology	3480
	A.1.6	Definitions and Concepts.....	3482
	A.1.7	Portability.....	3482
	A.2	Conformance.....	3483
	A.2.1	Implementation Conformance	3483
	A.2.2	Application Conformance.....	3487
	A.2.3	Language-Dependent Services for the C Programming Language	3487
	A.2.4	Other Language-Related Specifications.....	3488
	A.3	Definitions.....	3488
	A.4	General Concepts	3511
	A.4.1	Concurrent Execution.....	3511
	A.4.2	Default Initialization.....	3511
	A.4.3	Directory Protection.....	3511
	A.4.4	Extended Security Controls	3511
	A.4.5	File Access Permissions.....	3511
	A.4.6	File Hierarchy	3512
	A.4.7	Filenames.....	3512
	A.4.8	Filename Portability.....	3514
	A.4.9	File Times Update	3514
	A.4.10	Host and Network Byte Order	3514
	A.4.11	Measurement of Execution Time	3514
	A.4.12	Memory Synchronization	3515
	A.4.13	Pathname Resolution.....	3516
	A.4.14	Process ID Reuse	3518
	A.4.15	Scheduling Policy	3518
	A.4.16	Seconds Since the Epoch	3518
	A.4.17	Semaphore.....	3519
	A.4.18	Thread-Safety.....	3520
	A.4.19	Tracing	3520
	A.4.20	Treatment of Error Conditions for Mathematical Functions	3520
	A.4.21	Treatment of NaN Arguments for Mathematical Functions	3520
	A.4.22	Utility	3520
	A.4.23	Variable Assignment.....	3520
	A.5	File Format Notation	3520
	A.6	Character Set.....	3521
	A.6.1	Portable Character Set	3521
	A.6.2	Character Encoding	3522
	A.6.3	C Language Wide-Character Codes	3522

A.6.4	Character Set Description File.....	3522
A.7	Locale.....	3525
A.7.1	General.....	3525
A.7.2	POSIX Locale.....	3525
A.7.3	Locale De nition.....	3526
A.7.4	Locale De nition Grammar.....	3533
A.7.5	Locale De nition Example.....	3533
A.8	Environment Variables.....	3537
A.8.1	Environment Variable Definition.....	3537
A.8.2	Internationalization Variables.....	3537
A.8.3	Other Environment Variables.....	3538
A.9	Regular Expressions.....	3539
A.9.1	Regular Expression Definitions.....	3540
A.9.2	Regular Expression General Requirements.....	3541
A.9.3	Basic Regular Expressions.....	3542
A.9.4	Extended Regular Expressions.....	3545
A.9.5	Regular Expression Grammar.....	3546
A.10	Directory Structure and Devices.....	3547
A.10.1	Directory Structure and Files.....	3547
A.10.2	Output Devices and Terminal Types.....	3548
A.11	General Terminal Interface.....	3548
A.11.1	Interface Characteristics.....	3549
A.11.2	Parameters that Can be Set.....	3553
A.12	Utility Conventions.....	3554
A.12.1	Utility Argument Syntax.....	3554
A.12.2	Utility Syntax Guidelines.....	3555
A.13	Headers.....	3558
A.13.1	Format of Entries.....	3558
A.13.2	Removed Headers in Issue 7.....	3558
Part	B System Interfaces.....	3559
Appendix	B Rationale for System Interfaces.....	3561
B.1	Introduction.....	3561
B.1.1	Change History.....	3561
B.1.2	Relationship to Other Formal Standards.....	3564
B.1.3	Format of Entries.....	3564
B.2	General Information.....	3565
B.2.1	Use and Implementation of Interfaces.....	3565
B.2.2	The Compilation Environment.....	3566
B.2.3	Error Numbers.....	3571
B.2.4	Signal Concepts.....	3575
B.2.5	Standard I/O Streams.....	3585
B.2.6	STREAMS.....	3586
B.2.7	XSI Interprocess Communication.....	3586
B.2.8	Realtime.....	3587
B.2.9	Threads.....	3633
B.2.10	Sockets.....	3661
B.2.11	Tracing.....	3664
B.2.12	Data Types.....	3689
B.2.13	Status Information.....	3691

	B.2.14	File Descriptor Allocation	3691
	B.3	System Interfaces.....	3691
	B.3.1	System Interfaces Removed in this Version	3691
	B.3.2	System Interfaces Removed in the Previous Version.....	3694
	B.3.3	Examples for Spawn	3694
Part	C	Shell and Utilities	3705
Appendix	C	Rationale for Shell and Utilities.....	3707
	C.1	Introduction	3707
	C.1.1	Change History	3707
	C.1.2	Relationship to Other Documents	3708
	C.1.3	Utility Limits.....	3709
	C.1.4	Grammar Conventions.....	3712
	C.1.5	Utility Description Defaults.....	3712
	C.1.6	Considerations for Utilities in Support of Files of Arbitrary Size	3716
	C.1.7	Built-In Utilities	3716
	C.2	Shell Command Language	3718
	C.2.1	Shell Introduction.....	3718
	C.2.2	Quoting.....	3718
	C.2.3	Token Recognition.....	3720
	C.2.4	Reserved Words.....	3721
	C.2.5	Parameters and Variables.....	3721
	C.2.6	Word Expansions	3727
	C.2.7	Redirection	3735
	C.2.8	Exit Status and Errors	3737
	C.2.9	Shell Commands	3738
	C.2.10	Shell Grammar.....	3746
	C.2.11	Signals and Error Handling.....	3747
	C.2.12	Shell Execution Environment.....	3747
	C.2.13	Pattern Matching Notation	3748
	C.2.14	Special Built-In Utilities.....	3749
	C.3	Batch Environment Services and Utilities	3749
	C.3.1	Batch General Concepts	3753
	C.3.2	Batch Services	3755
	C.3.3	Common Behavior for Batch Environment Utilities.....	3756
	C.4	Utilities.....	3756
	C.4.1	Utilities Removed in this Version	3756
	C.4.2	Utilities Removed in the Previous Version.....	3756
	C.4.3	Exclusion of Utilities.....	3756
Part	D	Portability Considerations.....	3761
Appendix	D	Portability Considerations (Informative).....	3763
	D.1	User Requirements.....	3763
	D.1.1	Configuration Interrogation	3764
	D.1.2	Process Management	3764
	D.1.3	Access to Data.....	3764
	D.1.4	Access to the Environment	3764
	D.1.5	Access to Determinism and Performance	

		Enhancements.....	3764
D.1.6		Operating System-Dependent Profile	3765
D.1.7		I/O Interaction	3765
D.1.8		Internationalization Interaction	3765
D.1.9		C-Language Extensions.....	3765
D.1.10		Command Language	3765
D.1.11		Interactive Facilities	3765
D.1.12		Accomplish Multiple Tasks Simultaneously	3765
D.1.13		Complex Data Manipulation	3766
D.1.14		File Hierarchy Manipulation	3766
D.1.15		Locale Configuration	3766
D.1.16		Inter-User Communication.....	3766
D.1.17		System Environment	3766
D.1.18		Printing	3766
D.1.19		Software Development.....	3766
D.2		Portability Capabilities.....	3767
D.2.1		Configuration Interrogation	3767
D.2.2		Process Management	3768
D.2.3		Access to Data.....	3768
D.2.4		Access to the Environment	3769
D.2.5		Bounded (Realtime) Response	3770
D.2.6		Operating System-Dependent Profile	3770
D.2.7		I/O Interaction	3770
D.2.8		Internationalization Interaction	3771
D.2.9		C-Language Extensions.....	3771
D.2.10		Command Language	3771
D.2.11		Interactive Facilities	3772
D.2.12		Accomplish Multiple Tasks Simultaneously	3772
D.2.13		Complex Data Manipulation	3772
D.2.14		File Hierarchy Manipulation	3773
D.2.15		Locale Configuration	3773
D.2.16		Inter-User Communication.....	3773
D.2.17		System Environment	3774
D.2.18		Printing	3774
D.2.19		Software Development.....	3774
D.2.20		Future Growth	3774
D.3		Profiling Considerations.....	3775
D.3.1		Configuration Options	3775
D.3.2		Configuration Options (Shell and Utilities)	3775
D.3.3		Configurable Limits	3777
D.3.4		Configuration Options (System Interfaces).....	3777
D.3.5		Configurable Limits	3782
D.3.6		Optional Behavior	3785
Part	E	Subprofiling Considerations.....	3787
Appendix	E	Subprofiling Considerations (Informative).....	3789
	E.1	Subprofiling Option Groups.....	3789
		Index.....	3795

List of Figures

B-1	Example of a System with Typed Memory	3605
B-2	Trace System Overview: for Offline Analysis	3669
B-3	Trace System Overview: for Online Analysis	3670
B-4	Trace System Overview: States of a Trace Stream	3672
B-5	Trace Another Process	3682
B-6	Trace Name Space Overview: With Third-Party Library	3683

List of Tables

3-1	Job Control Job ID Formats.....	66
5-1	Escape Sequences and Associated Actions	121
6-1	Portable Character Set	125
6-2	Non-Portable Control Characters	130
7-1	Valid Character Class Combinations.....	142
10-1	Control Character Names	198
2-1	Value of Level for Socket Options.....	528
2-2	Socket-Level Options.....	529
2-3	Trace Option: System Trace Events.....	544
2-4	Trace and Trace Event Filter Options: System Trace Events.....	544
2-5	Trace and Trace Log Options: System Trace Events.....	545
2-6	Trace, Trace Log, and Trace Event Filter Options: System Trace Events	545
2-7	Trace Option: User Trace Event.....	546
1-1	Actions when Creating a File that Already Exists.....	2329
1-2	Selected ISO C Standard Operators and Control Flow Keywords	2332
1-3	Utility Limit Minimum Values	2333
1-4	Symbolic Utility Limits	2334
1-5	Regular Built-In Utilities	2344
3-1	Batch Utilities.....	2427
3-2	Environment Variable Summary	2431
3-3	Next State Table.....	2433
3-4	Results/Output Table	2435
3-5	Batch Services Summary	2442
A-1	Historical Practice for Symbolic Links.....	3507

Trademarks

The following information is given for the convenience of users of POSIX.1-2017 and does not constitute an endorsement by the IEEE or The Open Group of these products. Equivalent products may be used if they can be shown to lead to the same results.

There may be other products mentioned in the text that might be covered by trademark protection and readers are advised to verify them independently.

AIX[®] and IBM[®] are registered trademarks of International Business Machines Corporation.

ArchiMate[®], DirecNet[®], Making Standards Work[®], OpenPegasus[®], The Open Group[®], TOGAF[®], UNIX[®], UNIXWARE[®], X/Open[®], and The Open Brand X[®] logo are registered trademarks and Boundaryless Information Flow[™], Build with Integrity Buy with Confidence[™], Dependability Through Assuredness[™], EMMM[™], FACE[™], the FACE[™] logo, IT4IT[™], the IT4IT[™] logo, O-DEF[™], O-PAS[™], Open FAIR[™], Open Platform 3.0[™], Open Process Automation[™], Open Trusted Technology Provider[™], Platform 3.0[™], SOSA[™], the Open O[™] logo, and The Open Group Certification logo (Open O and check[™]) are trademarks of The Open Group.

AT&T[®] is a registered trademark of AT&T in the USA and other countries.

BSD[™] is a trademark of the University of California, Berkeley, USA.

Hewlett-Packard[®], HP[®], and HP-UX[®] are registered trademarks of Hewlett-Packard Company.

IEEE[®] and POSIX[®] are registered trademarks, and 754[™], 854[™], 1003.0[™], 1003.1[™], 1003.1d[™], 1003.1g[™], 1003.1j[™], 1003.1q[™], 1003.2[™], 1003.2a[™], 1003.2d[™], 1003.9[™], and 1003.13[™] are trademarks of The Institute of Electrical and Electronic Engineers, Inc.

Linux[®] is a registered trademark of Linus Torvalds.

Sun[®] and Sun Microsystems[®] are registered trademarks of Oracle America, Inc.

/usr/group[®] is a registered trademark of UniForum, the International Network of UNIX System Users.

Acknowledgements

The contributions of the following organizations to the development of POSIX.1-2017 are gratefully acknowledged:

AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation

Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems Inc. for permission to reproduce portions of their copyrighted documentation

ISO/IEC JTC 1/SC 22/WG 14 C Language Committee

Red Hat Inc. for permission to reproduce portions of its copyrighted documentation

POSIX.1-2017 was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

Referenced Documents

Normative References

Normative references for POSIX.1-2017 are defined in [Section 1.3](#) (on page 4).

Informative References

The following documents are referenced in POSIX.1-2017:

1984 /usr/group Standard

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

Almasi and Gottlieb

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

ANSI C

American National Standard for Information Systems: Standard X3.159-1989, Programming Language C.

ANSI X3.226-1994

American National Standard for Information Systems: Standard X3.226-1994, Programming Language Common LISP.

Brawer

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989, ISBN: 0-12-128470-0.

DeRemer and Pennello Article

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets* SigPlan Notices, Volume 15, No. 8, August 1979.

Draft ANSI X3J11.1

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

FIPS 151-1

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

FIPS 151-2

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

HP-UX Manual

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

IEC 60559: 1989

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously designated IEC 559: 1989).

IEEE Standards Terms

IEEE 100, The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition.

Referenced Documents

- IEEE Std 754™-1985
IEEE Std 754-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic.
- IEEE Std 854™-1987
IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.
- IEEE Std 1003.9™-1992
IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- IETF RFC 791
Internet Protocol, Version 4 (IPv4), September 1981 (available at: www.ietf.org/rfc/rfc0791.txt).
- IETF RFC 819
The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August 1982 (available at: www.ietf.org/rfc/rfc0819.txt).
- IETF RFC 822
Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982 (available at: www.ietf.org/rfc/rfc0822.txt).
- IETF RFC 919
Broadcasting Internet Datagrams, J. Mogul, October 1984 (available at: www.ietf.org/rfc/rfc0919.txt).
- IETF RFC 920
Domain Requirements, J. Postel, J. Reynolds, October 1984 (available at: www.ietf.org/rfc/rfc0920.txt).
- IETF RFC 921
Domain Name System Implementation Schedule, J. Postel, October 1984 (available at: www.ietf.org/rfc/rfc0921.txt).
- IETF RFC 922
Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984 (available at: www.ietf.org/rfc/rfc0922.txt).
- IETF RFC 1034
Domain Names ¶ Concepts and Facilities, PMockapetris, November 1987 (available at: www.ietf.org/rfc/rfc1034.txt).
- IETF RFC 1035
Domain Names ¶ Implementation and Specification, PMockapetris, November 1987 (available at: www.ietf.org/rfc/rfc1035.txt).
- IETF RFC 1123
Requirements for Internet Hosts ¶ Application and Support, R. Braden, October 1989 (available at: www.ietf.org/rfc/rfc1123.txt).
- IETF RFC 1886
DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson, December 1995 (available at: www.ietf.org/rfc/rfc1886.txt).
- IETF RFC 2045
Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies, N. Freed, N. Borenstein, November 1996 (available at: www.ietf.org/rfc/rfc2045.txt).

IETF RFC 2181

Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997 (available at: www.ietf.org/rfc/rfc2181.txt).

IETF RFC 2373

Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998 (available at: www.ietf.org/rfc/rfc2373.txt).

IETF RFC 2460

Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998 (available at: www.ietf.org/rfc/rfc2460.txt).

Internationalisation Guide

Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304), published by The Open Group.

ISO 2375: 1985

ISO 2375: 1985, Data Processing — Procedure for Registration of Escape Sequences.

ISO 8652: 1987

ISO 8652: 1987, Programming Languages — Ada (technically identical to ANSI standard 1815A-1983).

ISO/IEC 1539: 1991

ISO/IEC 1539: 1991, Information Technology — Programming Languages — Fortran (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).

ISO/IEC 4873: 1991

ISO/IEC 4873: 1991, Information Technology — ISO 8-bit Code for Information Interchange — Structure and Rules for Implementation.

ISO/IEC 6429: 1992

ISO/IEC 6429: 1992, Information Technology — Control Functions for Coded Character Sets.

ISO/IEC 6937: 1994

ISO/IEC 6937: 1994, Information Technology — Coded Graphic Character Set for Text Communication — Latin Alphabet.

ISO/IEC 8802-3: 1996

ISO/IEC 8802-3: 1996, Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.

ISO/IEC 8859

ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:

Part 1: Latin Alphabet No. 1

Part 2: Latin Alphabet No. 2

Part 3: Latin Alphabet No. 3

Part 4: Latin Alphabet No. 4

Part 5: Latin/Cyrillic Alphabet

Part 6: Latin/Arabic Alphabet

Part 7: Latin/Greek Alphabet

Part 8: Latin/Hebrew Alphabet

Part 9: Latin Alphabet No. 5

Part 10: Latin Alphabet No. 6

Part 11: Latin/Thai Alphabet

Referenced Documents

- Part 13: Latin Alphabet No. 7
Part 14: Latin Alphabet No. 8 (Celtic)
Part 15: Latin Alphabet No. 9
Part 16: Latin Alphabet No. 10
- ISO/IEC 9899:1990
ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).
- ISO POSIX-1:1996
ISO/IEC 9945-1:1996, Information Technology ‡ Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995.
- ISO POSIX-2:1993
ISO/IEC 9945-2:1993, Information Technology ‡ Portable Operating System Interface (POSIX) ‡ Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2™-1992, as amended by ANSI/IEEE Std 1003.2a™-1992).
- Issue 1
X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).
- Issue 2
X/Open Portability Guide, January 1987:
Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)
- Issue 3
X/Open Specification, 1988, 1989, February 1992:
Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- Issue 4
CAE Specification, July 1992, published by The Open Group:
System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)
System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)

Issue 4, Version 2

CAE Specification, August 1994, published by The Open Group:

System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)

Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)

System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)

Issue 5

Technical Standard, February 1997, published by The Open Group:

System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)

Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)

System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)

Issue 6

Technical Standard, April 2004, published by The Open Group:

Base Definitions (XBD), Issue 6 (ISBN: 1-931624-43-7, C046)

System Interfaces (XSH), Issue 6 (ISBN: 1-931624-44-5, C047)

Shell and Utilities (XCU), Issue 6 (ISBN: 1-931624-45-3, C048)

Knuth Article

Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control, Volume 8, No. 6, October 1965.

KornShell

Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming Language*, March 1995, Prentice Hall.

MSE Working Draft

Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31 March 1992.

POSIX.0: 1995

IEEE Std 1003.0™-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical to ISO/IEC TR 14252).

POSIX.1: 1988

IEEE Std 1003.1™-1988, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1: 1990

IEEE Std 1003.1™-1990, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language].

POSIX.1a

P1003.1a, Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) (C Language) Amendment.

POSIX.1d: 1999

IEEE Std 1003.1d™-1999, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) (C Language)

Referenced Documents

- Amendment 4: Additional Realtime Extensions [C Language].
- POSIX.1g: 2000
IEEE Std 1003.1g™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) ‡ Part 1: System Application Program Interface (API) ‡ Amendment 6: Protocol-Independent Interfaces (PII).
- POSIX.1j: 2000
IEEE Std 1003.1j™-2000, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) ‡ Part 1: System Application Program Interface (API) ‡ Amendment 5: Advanced Realtime Extensions [C Language].
- POSIX.1q: 2000
IEEE Std 1003.1q™-2000, IEEE Standard for Information Technology ‡ Portable Operating System Interface (POSIX) ‡ Part 1: System Application Program Interface (API) ‡ Amendment 7: Tracing [C Language].
- POSIX.2: 1992
IEEE Std 1003.2™-1992, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) ‡ Part 2: Shell and Utilities.
- POSIX.2b
P1003.2b, Standard for Information Technology ‡ Portable Operating System Interface (POSIX) ‡ Part 2: Shell and Utilities ‡ Amendment.
- POSIX.2d: 1994
IEEE Std 1003.2d™-1994, IEEE Standard for Information Technology — Portable Operating System Interface (POSIX) ‡ Part 2: Shell and Utilities ‡ Amendment 1: Batch Environment.
- POSIX.13: 1998
IEEE Std 1003.13™-1998, IEEE Standard for Information Technology — Standardized Application Environment Profile (AEP) — POSIX Realtime Application Support.
- Sarwate Article
Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications of the ACM, Volume 30, No. 8, August 1988.
- Sprunt, Sha, and Lehoczky
Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*, The Journal of Real-Time Systems, Volume 1, 1989, Pages 27-60.
- SVID, Issue 1
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 1; Morristown, NJ, UNIX Press, 1985.
- SVID, Issue 2
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 2; Morristown, NJ, UNIX Press, 1986.
- SVID, Issue 3
American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue 3; Morristown, NJ, UNIX Press, 1989.
- The AWK Programming Language
Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming Language*, Reading, MA, Addison-Wesley 1988.
- The C Programming Language
Kernighan, Brian W. and Ritchie, Dennis M., *The C Programming Language*, Englewood Cliffs, NJ, Prentice Hall, 1st Edition (February 1978) ISBN 0-13-110163-3; 2nd Edition (March

1988) ISBN 0-13-110362-8.

UNIX Programmer's Manual

American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January 1979.

XNS, Issue 4

CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438), published by The Open Group.

XNS, Issue 5

CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523), published by The Open Group.

XNS, Issue 5.2

Technical Standard, January 2000, Networking Services (XNS), Issue 5.2 (ISBN: 1-85912-241-8, C808), published by The Open Group.

X/Open Curses, Issue 4, Version 2

CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3, C610), published by The Open Group.

Yacc

Yacc: Yet Another Compiler Compiler, Stephen C. Johnson, 1978.

Source Documents

Parts of the following documents were used to create the base documents for POSIX.1-2001:

AIX 3.2 Manual

AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and Extensions, 1990, 1992 (Part No. SC23-2382-00).

OSF/1

OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).

OSF AES

Application Environment Specification (AES) Operating System Programming Interfaces Volume, Revision A (ISBN: 0-13-043522-8).

System V Release 2.0

‡ *UNIX System V Release 2.0 Programmer's Reference Manual* (April 1984 - Issue 2).

‡ *UNIX System V Release 2.0 Programming Guide* (April 1984 - Issue 2).

System V Release 4.2

Operating System API Reference, UNIX [®]SVR4.2 (1992) (ISBN: 0-13-017658-3).



1 **Vol. 1:**
2 **Base Definitions, Issue 7**

3 *The Open Group*
4 *The Institute of Electrical and Electronics Engineers, Inc.*

Introduction

1.1 Scope

POSIX.1-2017 defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. It is intended to be used by both application developers and system implementors.

POSIX.1-2017 comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of POSIX.1-2017, including utility conventions and C-language header definitions, are included in the Base Definitions volume of POSIX.1-2017.
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of POSIX.1-2017.
3. Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume of POSIX.1-2017.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of POSIX.1-2017 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of POSIX.1-2017.

The following areas are outside of the scope of POSIX.1-2017:

Graphics interfaces

Database management system interfaces

Record I/O considerations

Object or binary code portability

System configuration and resource availability

POSIX.1-2017 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

The facilities provided in POSIX.1-2017 are drawn from the following base documents:

IEEE Std 1003.1, 2004 Edition (POSIX-1) (incorporating IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002, and IEEE Std 1003.1-2001/Cor 2-2004)

40 The Open Group Technical Standard, 2006, Extended API Set Part 1
 41 The Open Group Technical Standard, 2006, Extended API Set Part 2
 42 The Open Group Technical Standard, 2006, Extended API Set Part 3
 43 The Open Group Technical Standard, 2006, Extended API Set Part 4
 44 ISO/IEC 9899:1999, Programming Languages — C, including ISO/IEC
 45 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC
 46 9899:1999/Cor.3

47 Emphasis has been placed on standardizing existing practice for existing users, with changes
 48 and additions limited to correcting deficiencies in the following areas:

49 Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1,
 50 and ISO/IEC defect reports against ISO/IEC 9945

51 Issues raised in corrigenda for The Open Group Technical Standards and working group
 52 resolutions from The Open Group

53 Issues arising from ISO TR 24715:2006, Conflicts between POSIX and the LSB

54 Changes to make the text self-consistent with the additional material merged

55 Features, marked Legacy or obsolescent in the base documents, have been considered for
 56 removal in this version

57 A review and reorganization of the options within the standard

58 Alignment with the ISO/IEC 9899:1999 standard, including ISO/IEC
 59 9899:1999/Cor.2:2004(E)

60 1.2 Conformance

61 Conformance requirements for POSIX.1-2017 are defined in [Chapter 2](#) (on page 15).

62 1.3 Normative References

63 The following standards contain provisions which, through references in POSIX.1-2017,
 64 constitute provisions of POSIX.1-2017. At the time of publication, the editions indicated were
 65 valid. All standards are subject to revision, and parties to agreements based on POSIX.1-2017 are
 66 encouraged to investigate the possibility of applying the most recent editions of the standards
 67 listed below. Members of IEC and ISO maintain registers of currently valid International
 68 Standards.

69 ANS X3.9-1978

70 (Reaffirmed 1989) American National Standard for Information Systems: Standard
 71 X3.9-1978, Programming Language FORTRAN.¹

72 ISO/IEC 646:1991

73 ISO/IEC 646:1991, Information Processing — ISO 7-Bit Coded Character Set for
 74 Information Interchange.²

75 1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY
 76 10018, USA.

- 77 ISO 4217: 2001
78 ISO 4217: 2001, Codes for the Representation of Currencies and Funds.
- 79 ISO 8601: 2004
80 ISO 8601: 2004, Data Elements and Interchange Formats — Information Interchange ‡
81 Representation of Dates and Times.
- 82 ISO C (1999)
83 ISO/IEC 9899: 1999, Programming Languages ‡ C, including ISO/IEC
84 9899: 1999/Cor.1: 2001(E), ISO/IEC 9899: 1999/Cor.2: 2004(E), and ISO/IEC
85 9899: 1999/Cor.3.
- 86 ISO/IEC 10646-1: 2000
87 ISO/IEC 10646-1: 2000, Information Technology — Universal Multiple-Octet Coded
88 Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

89 1.4 Change History

90 Change history is described in the Rationale (Informative) volume of POSIX.1-2017, and in the
91 CHANGE HISTORY section of reference pages.

92 1.5 Terminology

93 For the purposes of POSIX.1-2017, the following terminology definitions apply:

94 **can**

95 Describes a permissible optional feature or behavior available to the user or application. The
96 feature or behavior is mandatory for an implementation that conforms to POSIX.1-2017. An
97 application can rely on the existence of the feature or behavior.

98 **implementation-defined**

99 Describes a value or behavior that is not defined by POSIX.1-2017 but is selected by an
100 implementor. The value or behavior may vary among implementations that conform to
101 POSIX.1-2017. An application should not rely on the existence of the value or behavior. An
102 application that relies on such a value or behavior cannot be assured to be portable across
103 conforming implementations.

104 The implementor shall document such a value or behavior so that it can be used correctly
105 by an application.

106 **legacy**

107 Describes a feature or behavior that is being retained for compatibility with older
108 applications, but which has limitations which make it inappropriate for developing portable
109 applications. New applications should use alternative means of obtaining equivalent
110 functionality.

111 **may**

112 Describes a feature or behavior that is optional for an implementation that conforms to
113 POSIX.1-2017. An application should not rely on the existence of the feature or behavior. An
114 application that relies on such a feature or behavior cannot be assured to be portable across
115 conforming implementations.

116 2. ISO/IEC documents can be obtained from the ISO office: 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse

117 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

118 **shall**

119 For an implementation that conforms to POSIX.1-2017, describes a feature or behavior that
120 is mandatory. An application can rely on the existence of the feature or behavior.

121 For an application or user, describes a behavior that is mandatory.

122 **should**

123 For an implementation that conforms to POSIX.1-2017, describes a feature or behavior that
124 is recommended but not mandatory. An application should not rely on the existence of the
125 feature or behavior. An application that relies on such a feature or behavior cannot be
126 assured to be portable across conforming implementations.

127 For an application, describes a feature or behavior that is recommended programming
128 practice for optimum portability.

129 **undefined**

130 Describes the nature of a value or behavior not defined by POSIX.1-2017 which results from
131 use of an invalid program construct or invalid data input.

132 The value or behavior may vary among implementations that conform to POSIX.1-2017. An
133 application should not rely on the existence or validity of the value or behavior. An
134 application that relies on any particular value or behavior cannot be assured to be portable
135 across conforming implementations.

136 **unspecified**

137 Describes the nature of a value or behavior not specified by POSIX.1-2017 which results
138 from use of a valid program construct or valid data input.

139 The value or behavior may vary among implementations that conform to POSIX.1-2017. An
140 application should not rely on the existence or validity of the value or behavior. An
141 application that relies on any particular value or behavior cannot be assured to be portable
142 across conforming implementations.

143 **1.6 Definitions and Concepts**

144 Definitions and concepts are defined in [Chapter 3](#) (on page 33) and [Chapter 4](#) (on page 107).

145 **1.7 Portability**

146 Some of the utilities in the Shell and Utilities volume of POSIX.1-2017 and functions in the
147 System Interfaces volume of POSIX.1-2017 describe functionality that might not be fully portable
148 to systems meeting the requirements for POSIX conformance (see [Chapter 2](#), on page 15).

149 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
150 the margin identifies the nature of the option, extension, or warning (see [Section 1.7.1](#), on page
151 7). For maximum portability, an application should avoid such functionality.

152 Unless the primary task of a utility is to produce textual material on its standard output,
153 application developers should not rely on the format or content of any such material that may be
154 produced. Where the primary task *is* to provide such material, but the output format is
155 incompletely specified, the description is marked with the OF margin code and shading.
156 Application developers are warned not to expect that the output of such an interface on one

157 system is any guide to its behavior on another system.

158 **1.7.1 Codes**

159 The codes and their meanings are as follows. See also [Section 1.7.2](#) (on page 13).

160 ADV **Advisory Information**

161 The functionality described is optional. The functionality described is also an extension to the
162 ISO C standard.

163 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.
164 Where additional semantics apply to a function, the material is identified by use of the ADV
165 margin legend.

166 BE **Batch Environment Services and Utilities**

167 The functionality described is optional.

168 Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section.
169 Where additional semantics apply to a utility, the material is identified by use of the BE margin
170 legend.

171 CD **C-Language Development Utilities**

172 The functionality described is optional.

173 Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section.
174 Where additional semantics apply to a utility, the material is identified by use of the CD margin
175 legend.

176 CPT **Process CPU-Time Clocks**

177 The functionality described is optional. The functionality described is also an extension to the
178 ISO C standard.

179 Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section.
180 Where additional semantics apply to a function, the material is identified by use of the CPT
181 margin legend.

182 CX **Extension to the ISO C standard**

183 The functionality described is an extension to the ISO C standard. Application developers may
184 make use of an extension as it is supported on all POSIX.1-2017-conforming systems.

185 With each function or header from the ISO C standard, a statement to the effect that “any
186 conflict is unintentional” is included. That is intended to refer to a direct conflict. POSIX.1-2017
187 acts in part as a profile of the ISO C standard, and it may choose to further constrain behaviors
188 allowed to vary by the ISO C standard. Such limitations and other compatible differences are not
189 considered conflicts, even if a CX mark is missing. The markings are for information only.

190 Where additional semantics apply to a function or header, the material is identified by use of the
191 CX margin legend.

192 FD **FORTTRAN Development Utilities**

193 The functionality described is optional.

194 Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section.
195 Where additional semantics apply to a utility, the material is identified by use of the FD margin
196 legend.

197 FR **FORTTRAN Runtime Utilities**

198 The functionality described is optional.

199 Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section.

200		Where additional semantics apply to a utility, the material is identified by use of the FR margin legend.
201		
202	FSC	File Synchronization
203		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
204		
205		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
206		Where additional semantics apply to a function, the material is identified by use of the FSC margin legend.
207		
208	IP6	IPV6
209		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
210		
211		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
212		Where additional semantics apply to a function, the material is identified by use of the IP6 margin legend.
213		
214	MC1	Non-Robust Mutex Priority Protection or Non-Robust Mutex Priority Inheritance or Robust Mutex Priority Protection or Robust Mutex Priority Inheritance
215		
216		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
217		
218		This is a shorthand notation for combinations of multiple option codes.
219		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
220		Where additional semantics apply to a function, the material is identified by use of the MC1 margin legend.
221		
222		Refer to Section 1.7.2 (on page 13).
223	ML	Process Memory Locking
224		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
225		
226		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
227		Where additional semantics apply to a function, the material is identified by use of the ML margin legend.
228		
229	MLR	Range Memory Locking
230		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
231		
232		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
233		Where additional semantics apply to a function, the material is identified by use of the MLR margin legend.
234		
235	MON	Monotonic Clock
236		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
237		
238		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
239		Where additional semantics apply to a function, the material is identified by use of the MON margin legend.
240		
241	MSG	Message Passing
242		The functionality described is optional. The functionality described is also an extension to the ISO C standard.
243		
244		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.

245		Where additional semantics apply to a function, the material is identified by use of the MSG
246		margin legend.
247	MX	IEC 60559 Floating-Point
248		The functionality described is optional. The functionality described is mandated by the ISO C
249		standard only for implementations that define <code>__STDC_IEC_559__</code> .
250	MXX	IEC 60559 Floating-Point Extension
251		The functionality described is optional. The functionality described is part of the IEC 60559
252		Floating-Point option, but is an extension to the ISO C standard.
253	OB	Obsolescent
254		The functionality described may be removed in a future version of this volume of POSIX.1-2017.
255		Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use
256		obsolescent features.
257		Where applicable, the material is identified by use of the OB margin legend.
258	OF	Output Format Incompletely Specified
259		The functionality described is an XSI extension. The format of the output produced by the
260		utility is not fully specified. It is therefore not possible to post-process this output in a consistent
261		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
262		Where applicable, the material is identified by use of the OF margin legend.
263	OH	Optional Header
264		In the SYNOPSIS section of some interfaces in the System Interfaces volume of POSIX.1-2017 an
265		included header is marked as in the following example:
266	OH	<code>#include <sys/types.h></code>
267		<code>#include <fcntl.h></code>
268		<code>int open(const char *path, int oflag, ...);</code>
269		The OH margin legend indicates that the optional header defines constants that will be needed if
270		the function is called with certain flag arguments; thus it may be required for some of the
271		functionality described, but is not needed otherwise.
272	PIO	Prioritized Input and Output
273		The functionality described is optional. The functionality described is also an extension to the
274		ISO C standard.
275		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
276		Where additional semantics apply to a function, the material is identified by use of the PIO
277		margin legend.
278	PS	Process Scheduling
279		The functionality described is optional. The functionality described is also an extension to the
280		ISO C standard.
281		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
282		Where additional semantics apply to a function, the material is identified by use of the PS
283		margin legend.
284	RPI	Robust Mutex Priority Inheritance
285		The functionality described is optional. The functionality described is also an extension to the
286		ISO C standard.
287		Where applicable, functions are marked with the RPI margin legend in the SYNOPSIS section.
288		Where additional semantics apply to a function, the material is identified by use of the RPI
289		margin legend.

290	RPP	Robust Mutex Priority Protection
291		The functionality described is optional. The functionality described is also an extension to the
292		ISO C standard.
293		Where applicable, functions are marked with the RPP margin legend in the SYNOPSIS section.
294		Where additional semantics apply to a function, the material is identified by use of the RPP
295		margin legend.
296	RS	Raw Sockets
297		The functionality described is optional. The functionality described is also an extension to the
298		ISO C standard.
299		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
300		Where additional semantics apply to a function, the material is identified by use of the RS
301		margin legend.
302	SD	Software Development Utilities
303		The functionality described is optional.
304		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
305		Where additional semantics apply to a utility, the material is identified by use of the SD margin
306		legend.
307	SHM	Shared Memory Objects
308		The functionality described is optional. The functionality described is also an extension to the
309		ISO C standard.
310		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
311		Where additional semantics apply to a function, the material is identified by use of the SHM
312		margin legend.
313	SIO	Synchronized Input and Output
314		The functionality described is optional. The functionality described is also an extension to the
315		ISO C standard.
316		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
317		Where additional semantics apply to a function, the material is identified by use of the SIO
318		margin legend.
319	SPN	Spawn
320		The functionality described is optional. The functionality described is also an extension to the
321		ISO C standard.
322		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
323		Where additional semantics apply to a function, the material is identified by use of the SPN
324		margin legend.
325	SS	Process Sporadic Server
326		The functionality described is optional. The functionality described is also an extension to the
327		ISO C standard.
328		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
329		Where additional semantics apply to a function, the material is identified by use of the SS
330		margin legend.
331	TCT	Thread CPU-Time Clocks
332		The functionality described is optional. The functionality described is also an extension to the
333		ISO C standard.
334		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.

335		Where additional semantics apply to a function, the material is identified by use of the TCT
336		margin legend.
337	TEF	Trace Event Filter
338		The functionality described is optional. This functionality is dependent on support for the Trace
339		option. The functionality described is also an extension to the ISO C standard.
340		Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
341		Where additional semantics apply to a function, the material is identified by use of the TEF
342		margin legend.
343	TPI	Non-Robust Mutex Priority Inheritance
344		The functionality described is optional. The functionality described is also an extension to the
345		ISO C standard.
346		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
347		Where additional semantics apply to a function, the material is identified by use of the TPI
348		margin legend.
349	TPP	Non-Robust Mutex Priority Protection
350		The functionality described is optional. The functionality described is also an extension to the
351		ISO C standard.
352		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
353		Where additional semantics apply to a function, the material is identified by use of the TPP
354		margin legend.
355	TPS	Thread Execution Scheduling
356		The functionality described is optional. The functionality described is also an extension to the
357		ISO C standard.
358		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
359		Where additional semantics apply to a function, the material is identified by use of the TPS
360		margin legend.
361	TRC	Trace
362		The functionality described is optional. The functionality described is also an extension to the
363		ISO C standard.
364		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
365		Where additional semantics apply to a function, the material is identified by use of the TRC
366		margin legend.
367	TRI	Trace Inherit
368		The functionality described is optional. This functionality is dependent on support for the Trace
369		option. The functionality described is also an extension to the ISO C standard.
370		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
371		Where additional semantics apply to a function, the material is identified by use of the TRI
372		margin legend.
373	TRL	Trace Log
374		The functionality described is optional. This functionality is dependent on support for the Trace
375		option. The functionality described is also an extension to the ISO C standard.
376		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
377		Where additional semantics apply to a function, the material is identified by use of the TRL
378		margin legend.

379	TSA	Thread Stack Address Attribute
380		The functionality described is optional. The functionality described is also an extension to the
381		ISO C standard.
382		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
383		Where additional semantics apply to a function, the material is identified by use of the TSA
384		margin legend.
385	TSH	Thread Process-Shared Synchronization
386		The functionality described is optional. The functionality described is also an extension to the
387		ISO C standard.
388		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
389		Where additional semantics apply to a function, the material is identified by use of the TSH
390		margin legend.
391	TSP	Thread Sporadic Server
392		The functionality described is optional. The functionality described is also an extension to the
393		ISO C standard.
394		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
395		Where additional semantics apply to a function, the material is identified by use of the TSP
396		margin legend.
397	TSS	Thread Stack Size Attribute
398		The functionality described is optional. The functionality described is also an extension to the
399		ISO C standard.
400		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
401		Where additional semantics apply to a function, the material is identified by use of the TSS
402		margin legend.
403	TYM	Typed Memory Objects
404		The functionality described is optional. The functionality described is also an extension to the
405		ISO C standard.
406		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
407		Where additional semantics apply to a function, the material is identified by use of the TYM
408		margin legend.
409	UP	User Portability Utilities
410		The functionality described is optional.
411		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
412		Where additional semantics apply to a utility, the material is identified by use of the UP margin
413		legend.
414	UU	UUCP Utilities
415		The functionality described is optional. The functionality described is also an extension to the
416		ISO C standard.
417		Where applicable, functions are marked with the UU margin legend in the SYNOPSIS section.
418		Where additional semantics apply to a function, the material is identified by use of the UU
419		margin legend.
420	XSI	X/Open System Interfaces
421		The functionality described is part of the X/Open Systems Interfaces option. Functionality
422		marked XSI is an extension to the ISO C standard. Application developers may confidently
423		make use of such extensions on all systems supporting the X/Open System Interfaces option.

424 If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that
 425 reference page is an extension. See [Section 2.1.4](#) (on page 19).

426 XSR **XSI STREAMS**

427 The functionality described is optional. The functionality described is also an extension to the
 428 ISO C standard.

429 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
 430 Where additional semantics apply to a function, the material is identified by use of the XSR
 431 margin legend.

432 **1.7.2 Margin Code Notation**

433 Some of the functionality described in POSIX.1-2017 depends on support of more than one
 434 option, or independently may depend on several options. The following notation for margin
 435 codes is used to denote the following cases.

436 **A Feature Dependent on One or Two Options**

437 In this case, margin codes have a <space> separator; for example:

438 SHM This feature requires support for only the Shared Memory Objects option.

439 SHM TYM This feature requires support for both the Shared Memory Objects option and the Typed
 440 Memory Objects option; that is, an application which uses this feature is portable only between
 441 implementations that provide both options.

442 **A Feature Dependent on Either of the Options Denoted**

443 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

444 SHM | TYM This feature is dependent on support for either the Shared Memory Objects option or the Typed
 445 Memory Objects option; that is, an application which uses this feature is portable between
 446 implementations that provide any (or all) of the options.

447 **A Feature Dependent on More than Two Options**

448 The following shorthand notations are used:

449 MC1 The MC1 margin code is shorthand for TPP | TPI | RPP | RPI. Features which are shaded with this
 450 margin code require support of either the Non-Robust Mutex Priority Protection option or the
 451 Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or
 452 the Robust Mutex Priority Inheritance option.

453 **Large Sections Dependent on an Option**

454 Where large sections of text are dependent on support for an option, a lead-in text block is
 455 provided and shaded accordingly; for example:

456 XSI This section describes extensions to support interprocess communication. The functionality
 457 described in this section shall be provided on implementations that support the XSI option (and
 458 the rest of this section is not further shaded).

Conformance

2.1 Implementation Conformance

For the purposes of POSIX.1-2017, the implementation conformance requirements given in this section apply.

2.1.1 Requirements

A conforming implementation shall meet all of the following criteria:

1. The system shall support all utilities, functions, and facilities defined within POSIX.1-2017 that are required for POSIX conformance (see [Section 2.1.3](#), on page 17). These interfaces shall support the functional behavior described herein.
2. The system may support the X/Open System Interfaces (XSI) option as described in [Section 2.1.4](#) (on page 19).
3. The system may support one or more options as described under [Section 2.1.5](#) (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
4. The system may provide non-standard extensions. These are features not required by POSIX.1-2017 and may include, but are not limited to:
 - ‡ additional functions
 - ‡ additional headers
 - ‡ additional symbols in standard headers
 - ‡ additional utilities
 - ‡ additional options for standard utilities
 - ‡ additional environment variables
 - ‡ additional file types
 - ‡ nonconforming file systems (for example, legacy file systems for which `_POSIX_NO_TRUNC` is false, case-insensitive file systems, or network file systems)
 - ‡ dynamically populated file systems (for example, `/proc`)
 - ‡ additional character special files with special properties (for example, `/dev/stdin`, `/dev/stdout`, and `/dev/stderr`)

Non-standard extensions of the utilities, functions, or facilities specified in POSIX.1-2017 should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by POSIX.1-2017. The conformance document shall define an environment in which an application can be run with the behavior specified by POSIX.1-2017. In no case shall such

493 an environment require modification of a Strictly Conforming POSIX Application (see
494 [Section 2.2.1](#), on page 29).

495 **Note:** If the documented method of setting up a conforming environment includes the need to set one
496 or more environment variables, then the values of those environment variables cannot include
497 any <space> characters, since the *confstr()* function must be able to return them in a
498 <space>-separated list of variable=value pairs. See XSH *confstr()* (on page 698).

499 2.1.2 Documentation

500 A conformance document with the following information shall be available for an
501 implementation claiming conformance to POSIX.1-2017. The conformance document shall have
502 the same structure as POSIX.1-2017, with the information presented in the appropriate sections
503 and subsections. Sections and subsections that consist solely of subordinate section titles, with
504 no other information, are not required. The conformance document shall not contain
505 information about extended facilities or capabilities outside the scope of POSIX.1-2017.

506 The conformance document shall contain a statement that indicates the full name, number, and
507 date of the standard that applies. The conformance document may also list international
508 software standards that are available for use by a Conforming POSIX Application. Applicable
509 characteristics where documentation is required by one of these standards, or by standards of
510 government bodies, may also be included.

511 The conformance document shall describe the limit values found in the headers [<limits.h>](#) (on
512 page 270) and [<unistd.h>](#) (on page 434), stating values, the conditions under which those values
513 may change, and the limits of such variations, if any.

514 The conformance document shall describe the behavior of the implementation for all
515 implementation-defined features defined in POSIX.1-2017. This requirement shall be met by
516 listing these features and providing either a specific reference to the system documentation or
517 providing full syntax and semantics of these features. When the value or behavior in the
518 implementation is designed to be variable or customized on each instantiation of the system, the
519 implementation provider shall document the nature and permissible ranges of this variation.

520 The conformance document may specify the behavior of the implementation for those features
521 where POSIX.1-2017 states that implementations may vary or where features are identified as
522 undefined or unspecified.

523 The conformance document shall not contain documentation other than that specified in the
524 preceding paragraphs except where such documentation is specifically allowed or required by
525 other provisions of POSIX.1-2017.

526 The phrases “shall document” or “shall be documented” in POSIX.1-2017 mean that
527 documentation of the feature shall appear in the conformance document, as described
528 previously, unless there is an explicit reference in the conformance document to show where the
529 information can be found in the system documentation.

530 The system documentation should also contain the information found in the conformance
531 document.

532 **2.1.3 POSIX Conformance**

533 A conforming implementation shall meet the following criteria for POSIX conformance.

534 **2.1.3.1 POSIX System Interfaces**

535 The following requirements apply to the system interfaces (functions and headers):

536 The system shall support all the mandatory functions and headers defined in
537 POSIX.1-2017, and shall set the symbolic constant `_POSIX_VERSION` to the value 200809L.538 Although all implementations conforming to POSIX.1-2017 support all the features
539 described below, there may be system-dependent or file system-dependent configuration
540 procedures that can remove or modify any or all of these features. Such configurations
541 should not be made if strict compliance is required.542 The following symbolic constants shall be defined with a value other than `-1`. If a constant
543 is defined with the value zero, applications should use the `sysconf()`, `pathconf()`, or
544 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the
545 system at that time or for the particular pathname in question.546 `‡_POSIX_CHOWN_RESTRICTED`547 The use of `chown()` is restricted to a process with appropriate privileges, and to
548 changing the group ID of a file only to the effective group ID of the process or to one
549 of its supplementary group IDs.550 `‡_POSIX_NO_TRUNC`551 Pathname components longer than `{NAME_MAX}` generate an error.

552 The following symbolic constants shall be defined by the implementation as follows:

553 `‡_POSIX2_C_BIND` Symbolic constants defined with the value 200809L:554 `_POSIX_ASYNCHRONOUS_IO`
555 `_POSIX_BARRIERS`
556 `_POSIX_CLOCK_SELECTION`
557 `_POSIX_MAPPED_FILES`
558 `_POSIX_MEMORY_PROTECTION`
559 `_POSIX_READER_WRITER_LOCKS`
560 `_POSIX_REALTIME_SIGNALS`
561 `_POSIX_SEMAPHORES`
562 `_POSIX_SPIN_LOCKS`
563 `_POSIX_THREAD_SAFE_FUNCTIONS`
564 `_POSIX_THREADS`
565 `_POSIX_TIMEOUTS`
566 `_POSIX_TIMERS`
567 `_POSIX2_C_BIND`568 `‡_POSIX2_C_BIND` Symbolic constants defined with a value greater than zero:569 `_POSIX_JOB_CONTROL`
570 `_POSIX_REGEX`
571 `_POSIX_SAVED_IDS`
572 `_POSIX_SHELL`

573 symbolic constants defined with a value other than `-1`.

574 `_POSIX_VDISABLE`

575 **Note:** The symbols above represent historical options that are no longer allowed as options, but
576 are retained here for backwards-compatibility of applications.

577 The system may support one or more options (see [Section 2.1.6](#), on page 26) denoted by the
578 following symbolic constants:

579 `_POSIX_ADVISORY_INFO`
580 `_POSIX_CPUTIME`
581 `_POSIX_FSYNC`
582 `_POSIX_IPV6`
583 `_POSIX_MEMLOCK`
584 `_POSIX_MEMLOCK_RANGE`
585 `_POSIX_MESSAGE_PASSING`
586 `_POSIX_MONOTONIC_CLOCK`
587 `_POSIX_PRIORITIZED_IO`
588 `_POSIX_PRIORITY_SCHEDULING`
589 `_POSIX_RAW_SOCKETS`
590 `_POSIX_SHARED_MEMORY_OBJECTS`
591 `_POSIX_SPAWN`
592 `_POSIX_SPORADIC_SERVER`
593 `_POSIX_SYNCHRONIZED_IO`
594 `_POSIX_THREAD_ATTR_STACKADDR`
595 `_POSIX_THREAD_CPUTIME`
596 `_POSIX_THREAD_ATTR_STACKSIZE`
597 `_POSIX_THREAD_PRIO_INHERIT`
598 `_POSIX_THREAD_PRIO_PROTECT`
599 `_POSIX_THREAD_PRIORITY_SCHEDULING`
600 `_POSIX_THREAD_PROCESS_SHARED`
601 `_POSIX_THREAD_SPORADIC_SERVER`
602 `_POSIX_TRACE`
603 `_POSIX_TRACE_EVENT_FILTER`
604 `_POSIX_TRACE_INHERIT`
605 `_POSIX_TRACE_LOG`
606 `_POSIX_TYPED_MEMORY_OBJECTS`
607 `_XOPEN_CRYPT`
608 `_XOPEN_REALTIME`
609 `_XOPEN_REALTIME_THREADS`
610 `_XOPEN_STREAMS`
611 `_XOPEN_UNIX`

612 If any of the symbolic constants `_POSIX_TRACE_EVENT_FILTER`, `_POSIX_TRACE_LOG`,
613 or `_POSIX_TRACE_INHERIT` is defined to have a value other than `-1`, then the symbolic
614 constant `_POSIX_TRACE` shall also be defined to have a value other than `-1`.

615 If the Advisory Information option is supported, there shall be at least one file system that
616 supports the functionality.

617 2.1.3.2 *POSIX Shell and Utilities*

618 The following requirements apply to the shell and utilities:

619 The system shall provide all the mandatory utilities in the Shell and Utilities volume of
620 POSIX.1-2017 with all the functional behavior described therein.621 The system shall support the Large File capabilities described in the Shell and Utilities
622 volume of POSIX.1-2017.623 The system may support one or more options (see [Section 2.1.6](#), on page 26) denoted by the
624 following symbolic constants. (The literal names below apply to the *getconf* utility.)

625 POSIX2_C_DEV
 626 POSIX2_CHAR_TERM
 627 POSIX2_FORT_DEV
 628 POSIX2_FORT_RUN
 629 POSIX2_LOCALEDEF
 630 POSIX2_PBS
 631 POSIX2_PBS_ACCOUNTING
 632 POSIX2_PBS_LOCATE
 633 POSIX2_PBS_MESSAGE
 634 POSIX2_PBS_TRACK
 635 POSIX2_SW_DEV
 636 POSIX2_UPE
 637 XOPEN_UNIX
 638 XOPEN_UUCP

639 Additional language bindings and development utility options may be provided in other related
 640 standards or in a future version of this standard. In the former case, additional symbolic
 641 constants of the same general form as shown in this subsection should be defined by the related
 642 standard document and made available to the application without requiring POSIX.1-2017 to be
 643 updated.

644 **2.1.4 XSI Conformance**

645 XSI This section describes the criteria for implementations providing conformance to the X/Open
 646 System Interfaces (XSI) option (see [Section 3.450](#), on page 105). The functionality described in
 647 this section shall be provided on implementations that support the XSI option (and the rest of
 648 this section is not further shaded).

649 POSIX.1-2017 describes utilities, functions, and facilities offered to application programs by the
 650 X/Open System Interfaces (XSI) option. An XSI-conforming implementation shall meet the
 651 criteria for POSIX conformance and the following requirements listed in this section.

652 XSI-conforming implementations shall set the symbolic constant `_XOPEN_UNIX` to a value
 653 other than `-1` and shall set the symbolic constant `_XOPEN_VERSION` to the value `700`.

654 2.1.4.1 *XSI System Interfaces*

655 The following requirements apply to the system interfaces when the XSI option is supported:

656 The system shall support all the functions and headers defined in POSIX.1-2017 as part of
 657 the XSI option denoted by the XSI marking in the SYNOPSIS section, and any extensions
 658 marked with the XSI option marking (see [Section 1.7.1](#), on page 7) within the text.

659 The system shall support the following options defined within POSIX.1-2017 (see [Section](#)
660 [2.1.6](#), on page 26):

```
661     _POSIX_FSYNC
662     _POSIX_THREAD_ATTR_STACKADDR
663     _POSIX_THREAD_ATTR_STACKSIZE
664     _POSIX_THREAD_PROCESS_SHARED
```

665 The system may support the following XSI Option Groups (see [Section 2.1.5.2](#), on page 22)
666 defined within POSIX.1-2017:

```
667     ‡ encryption
668     ‡ realtime
669     ‡ advanced Realtime
670     ‡ realtime Threads
671     ‡ advanced Realtime Threads
672     ‡ threading
673     ‡ STREAMS
```

674 2.1.4.2 XSI Shell and Utilities Conformance

675 The following requirements apply to the shell and utilities when the XSI option is supported:

676 The system shall support all the utilities defined in the Shell and Utilities volume of
677 POSIX.1-2017 as part of the XSI option denoted by the XSI marking in the SYNOPSIS
678 section, and any extensions marked with the XSI option marking (see [Section 1.7.1](#), on
679 page 7) within the text.

680 The system shall support the User Portability Utilities option and the Terminal
681 Characteristics option.

682 The system shall support creation of locales (see [Chapter 7](#), on page 135).

683 The C-language Development utility *c99* shall be supported.

684 The XSI Development Utilities option may be supported. It consists of the following
685 software development utilities:

```
686     admin  delta  rmdel  val
687     cflow  get    sact   what
688     ctags  nm     sccs
689     cxref  prs   unget
```

690 2.1.5 Option Groups

691 An Option Group is a group of related functions or options defined within the System Interfaces
692 volume of POSIX.1-2017.

693 If an implementation supports an Option Group, then the system shall support the functional
694 behavior described herein.

695 If an implementation does not support an Option Group, then the system need not support the
696 functional behavior described herein.

697 2.1.5.1 Subprofiling Considerations

698 Profiling standards supporting functional requirements less than that required in POSIX.1-2017
 699 may subset both mandatory and optional functionality required for POSIX Conformance (see
 700 [Section 2.1.3](#), on page 17) or XSI Conformance (see [Section 2.1.4](#), on page 19). Such profiles shall
 701 organize the subsets into Subprofiling Option Groups.

702 XRAT [Appendix E](#) (on page 3789) describes a representative set of such Subprofiling Option
 703 Groups for use by profiles applicable to specialized realtime systems. POSIX.1-2017 does not
 704 require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols
 705 defined in any header) or at runtime (via *sysconf()* or *getconf*).

706 A Subprofiling Option Group may provide basic system functionality that other Subprofiling
 707 Option Groups and other options depend upon.³ If a profile of POSIX.1-2017 does not require an
 708 implementation to provide a Subprofiling Option Group that provides features utilized by a
 709 required Subprofiling Option Group (or option),⁴ the profile shall specify⁵ all of the following:

710 Restricted or altered behavior of interfaces defined in POSIX.1-2017 that may differ on an
 711 implementation of the profile

712 Additional behaviors that may produce undefined or unspecified results

713 Additional implementation-defined behavior that implementations shall be required to
 714 document in the profile's conformance document

715 if any of the above is a result of the profile not requiring an interface required by POSIX.1-2017.

716 The following additional rules shall apply to all profiles of POSIX.1-2017:

717 Any application that conforms to that profile shall also conform to POSIX.1-2017, unless
 718 the application depends on the definition of a profile support indicator macro in
 719 <unistd.h> (that is, a profile shall not require restricted, altered, or extended behaviors of
 720 an implementation of POSIX.1-2017).

721 Profiles are permitted to require the definition of a *profile support indicator macro* with a
 722 name beginning `_POSIX_AEP_` in <unistd.h>.

723 Profiles shall require the definition of the macro `_POSIX_SUBPROFILE` in <unistd.h> on
 724 implementations that do not meet all of the requirements of a POSIX.1-conforming
 725 implementation.

726 Profiles are permitted to add additional requirements to the limits defined in <limits.h>
 727 and <stdint.h>, subject to the following:

728 For the limits in <limits.h> and <stdint.h>:

729 3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are
 730 needed by any interface in POSIX.1-2017 that parses a *path* argument. If a profile requires support for the Device Input and Output
 731 profiling option group but does not require support for the File System profiling option group, the profile must specify how pathname
 732 resolution is to behave in that profile, how the `O_CREAT` flag to *open()* is to be handled (and the use of the character 'a' in the *mode*
 733 argument of *fopen()* when a pathname argument names a file that does not exist), and specify lots of other details.

734 4. As an example, POSIX.1-2017 requires that implementations claiming to support the Range Memory Locking option also support the
 735 Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that
 736 the Process Memory Locking option be supplied as long as the profile specifies everything an application developer or system implementor
 737 would have to know to build an application or implementation conforming to the profile.

738 5. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified
 739 results.

740 ¶ If the limit is specified as having a fixed value, it shall not be changed by a profile.
 741 ¶ If a limit is specified as having a minimum or maximum acceptable value, it may be
 742 changed by a profile as follows:
 743 ‡ A profile may increase a minimum acceptable value, but shall not make a
 744 minimum acceptable value smaller.
 745 ‡ A profile may reduce a maximum acceptable value, but shall not make a
 746 maximum acceptable value larger.
 747 A profile shall not change a limit specified as having a minimum or maximum value into a
 748 limit specified as having a fixed value.
 749 A profile shall not create new limits.
 750 Any implementation that conforms to POSIX.1-2017 (including all options and extended
 751 limits required by the profile) shall also conform to that profile, except for the possible
 752 omission from `<unistd.h>` of a profile support indicator macro required by the profile.

753 2.1.5.2 XSI Option Groups

754 XSI This section describes Option Groups to support the definition of XSI conformance within the
 755 System Interfaces volume of POSIX.1-2017. The functionality described in this section shall be
 756 provided on implementations that support the XSI option and the appropriate Option Group
 757 (and the rest of this section is not further shaded).

758 The following Option Groups are defined.

759 **Encryption**

760 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes
 761 the following functions:

762 `crypt()`, `encrypt()`, `setkey()`

763 These functions are marked CRYPT.

764 Due to export restrictions on the decoding algorithm in some countries, implementations may
 765 be restricted in making these functions available. All the functions in the Encryption Option
 766 Group may therefore return `[ENOSYS]` or, alternatively, `encrypt()` shall return `[ENOSYS]` for the
 767 decryption operation.

768 An implementation that claims conformance to this Option Group shall set `_XOPEN_CRYPT` to
 769 a value other than `-1`.

770 **Realtime**

771 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

772 This Option Group includes a set of realtime functions drawn from options within POSIX.1-2017
 773 (see [Section 2.1.6](#), on page 26).

774 Where entire functions are included in the Option Group, the NAME section is marked with
 775 REALTIME. Where additional semantics have been added to existing pages, the new material is
 776 identified by use of the appropriate margin legend for the underlying option defined within
 777 POSIX.1-2017.

778 An implementation that claims conformance to this Option Group shall set
 779 `_XOPEN_REALTIME` to a value other than `-1`.

780 This Option Group consists of the set of the following options from within POSIX.1-2017 (see
781 [Section 2.1.6](#), on page 26):

```
782     _POSIX_FSYNC
783     _POSIX_MEMLOCK
784     _POSIX_MEMLOCK_RANGE
785     _POSIX_MESSAGE_PASSING
786     _POSIX_PRIORITIZED_IO
787     _POSIX_PRIORITY_SCHEDULING
788     _POSIX_SHARED_MEMORY_OBJECTS
789     _POSIX_SYNCHRONIZED_IO
```

790 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the
791 following symbolic constants shall be defined by the implementation to have the value 200809L:

```
792     _POSIX_MEMLOCK
793     _POSIX_MEMLOCK_RANGE
794     _POSIX_MESSAGE_PASSING
795     _POSIX_PRIORITY_SCHEDULING
796     _POSIX_SHARED_MEMORY_OBJECTS
797     _POSIX_SYNCHRONIZED_IO
```

798 The functionality associated with `_POSIX_FSYNC` shall always be supported on XSI-conformant
799 systems.

800 Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If
801 `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by
802 `aio_read()`, `aio_write()`, and `lio_listio()` shall be submitted at a priority equal to the scheduling
803 priority equal to a base scheduling priority minus `aiocbp` ~~`aio_reqprio`~~. If Thread Execution
804 Scheduling is not supported, then the base scheduling priority is that of the calling process;
805 otherwise, the base scheduling priority is that of the calling thread. The implementation shall
806 also document for which files I/O prioritization is supported.

807 **Advanced Realtime**

808 An implementation that claims conformance to this Option Group shall also support the
809 Realtime Option Group.

810 Where entire functions are included in the Option Group, the NAME section is marked with
811 ADVANCED REALTIME. Where additional semantics have been added to existing pages, the
812 new material is identified by use of the appropriate margin legend for the underlying option
813 defined within POSIX.1-2017.

814 This Option Group consists of the set of the following options from within POSIX.1-2017 (see
815 [Section 2.1.6](#), on page 26):

```
816     _POSIX_ADVISORY_INFO
817     _POSIX_CPUTIME
818     _POSIX_MONOTONIC_CLOCK
819     _POSIX_SPAWN
820     _POSIX_SPORADIC_SERVER
821     _POSIX_TYPED_MEMORY_OBJECTS
```

822 If the implementation supports the Advanced Realtime Option Group, then the following
823 symbolic constants shall be defined by the implementation to have the value 200809L:

824 _POSIX_ADVISORY_INFO
 825 _POSIX_CPUTIME
 826 _POSIX_MONOTONIC_CLOCK
 827 _POSIX_SPAWN
 828 _POSIX_SPORADIC_SERVER
 829 _POSIX_TYPED_MEMORY_OBJECTS

830 If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant
 831 `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the
 832 value 200809L.

833 **Realtime Threads**

834 The Realtime Threads Option Group is denoted by the symbolic constant
 835 `_XOPEN_REALTIME_THREADS`.

836 This Option Group consists of the set of the following options from within POSIX.1-2017 (see
 837 [Section 2.1.6](#), on page 26):

838 _POSIX_THREAD_PRIO_INHERIT
 839 _POSIX_THREAD_PRIO_PROTECT
 840 _POSIX_THREAD_PRIORITY_SCHEDULING
 841 _POSIX_THREAD_ROBUST_PRIO_INHERIT
 842 _POSIX_THREAD_ROBUST_PRIO_PROTECT

843 Where applicable, whole pages are marked `REALTIME THREADS`, together with the
 844 appropriate option margin legend for the SYNOPSIS section (see [Section 1.7.1](#), on page 7).

845 An implementation that claims conformance to this Option Group shall set
 846 `_XOPEN_REALTIME_THREADS` to a value other than `-1`.

847 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the
 848 following options shall also be defined by the implementation to have the value 200809L:

849 _POSIX_THREAD_PRIO_INHERIT
 850 _POSIX_THREAD_PRIO_PROTECT
 851 _POSIX_THREAD_PRIORITY_SCHEDULING
 852 _POSIX_THREAD_ROBUST_PRIO_INHERIT
 853 _POSIX_THREAD_ROBUST_PRIO_PROTECT

854 **Advanced Realtime Threads**

855 An implementation that claims conformance to this Option Group shall also support the
 856 Realtime Threads Option Group.

857 Where entire functions are included in the Option Group, the NAME section is marked with
 858 `ADVANCED REALTIME THREADS`. Where additional semantics have been added to existing
 859 pages, the new material is identified by use of the appropriate margin legend for the underlying
 860 option defined within POSIX.1-2017.

861 This Option Group consists of the set of the following options from within POSIX.1-2017 (see
 862 [Section 2.1.6](#), on page 26):

863 _POSIX_THREAD_CPUTIME
 864 _POSIX_THREAD_SPORADIC_SERVER

865 If the symbolic constant `_POSIX_THREAD_SPORADIC_SERVER` is defined to have the value

866 200809L, then the symbolic constant `_POSIX_THREAD_PRIORITY_SCHEDULING` shall also be
867 defined by the implementation to have the value 200809L.

868 If the implementation supports the Advanced Realtime Threads Option Group, then the
869 following symbolic constants shall be defined by the implementation to have the value 200809L:

870 `_POSIX_THREAD_CPUTIME`
871 `_POSIX_THREAD_SPORADIC_SERVER`

872 **Tracing**

873 This Option Group includes a set of tracing functions drawn from options within POSIX.1-2017
874 (see [Section 2.1.6](#), on page 26).

875 Where entire functions are included in the Option Group, the NAME section is marked with
876 TRACING. Where additional semantics have been added to existing pages, the new material is
877 identified by use of the appropriate margin legend for the underlying option defined within
878 POSIX.1-2017.

879 This Option Group consists of the set of the following options from within POSIX.1-2017 (see
880 [Section 2.1.6](#), on page 26):

881 `_POSIX_TRACE`
882 `_POSIX_TRACE_EVENT_FILTER`
883 `_POSIX_TRACE_LOG`
884 `_POSIX_TRACE_INHERIT`

885 If the implementation supports the Tracing Option Group, then the following symbolic
886 constants shall be defined by the implementation to have the value 200809L:

887 `_POSIX_TRACE`
888 `_POSIX_TRACE_EVENT_FILTER`
889 `_POSIX_TRACE_LOG`
890 `_POSIX_TRACE_INHERIT`

891 **XSI STREAMS**

892 OB XSR This section describes the XSI STREAMS Option Group, denoted by the symbolic constant
893 `_XOPEN_STREAMS`. The functionality described in this section shall be provided on
894 implementations that support the XSI STREAMS option (and the rest of this section is not
895 further shaded).

896 This Option Group includes functionality related to STREAMS, a uniform mechanism for
897 implementing networking services and other character-based I/O as described in XSH [Section](#)
898 [2.6](#) (on page 500).

899 It includes the following functions:

900 `fattach()` `ioctl()`
901 `fdetach()` `isastream()`
902 `getmsg()` `putmsg()`
903 `getpmsg()` `putpmsg()`

904 and the `<stropts.h>` header.

905 Where applicable, whole pages are marked STREAMS, together with the appropriate option
906 margin legend for the SYNOPSIS section (see [Section 1.7.1](#), on page 7). Where additional
907 semantics have been added to existing pages, the new material is identified by use of the

908 appropriate margin legend for the underlying option defined within POSIX.1-2017.

909 An implementation that claims conformance to this Option Group shall set `_XOPEN_STREAMS`
 910 to a value other than `-1`.

911 2.1.6 Options

912 The symbolic constants defined in `<unistd.h>`, [Constants for Options and Option Groups](#) (on
 913 page 434) reflect implementation options for POSIX.1-2017. These symbols can be used by the
 914 application to determine which of three categories of support for optional facilities are provided
 915 by the implementation.

916 1. Option not supported for compilation.

917 The implementation advertises at compile time (by defining the constant in `<unistd.h>`
 918 with value `-1`, or by leaving it undefined) that the option is not supported for compilation
 919 and, at the time of compilation, is not supported for runtime use. In this case, the headers,
 920 data types, function interfaces, and utilities required only for the option need not be
 921 present. A later runtime check using the `fpathconf()`, `pathconf()`, or `sysconf` functions
 922 defined in the System Interfaces volume of POSIX.1-2017 or the `getconf` utility defined in
 923 the Shell and Utilities volume of POSIX.1-2017 can in some circumstances indicate that
 924 the option is supported at runtime. (For example, an old application binary might be run
 925 on a newer implementation to which support for the option has been added.)

926 2. Option always supported.

927 The implementation advertises at compile time (by defining the constant in `<unistd.h>`
 928 with a value greater than zero) that the option is supported both for compilation and for
 929 use at runtime. In this case, all headers, data types, function interfaces, and utilities
 930 required only for the option shall be available and shall operate as specified. Runtime
 931 checks with `fpathconf()`, `pathconf()`, or `sysconf` shall indicate that the option is supported.

932 3. Option might or might not be supported at runtime.

933 The implementation advertises at compile time (by defining the constant in `<unistd.h>`
 934 with value zero) that the option is supported for compilation and might or might not be
 935 supported at runtime. In this case, the `fpathconf()`, `pathconf()`, or `sysconf()` functions
 936 defined in the System Interfaces volume of POSIX.1-2017 or the `getconf` utility defined in
 937 the Shell and Utilities volume of POSIX.1-2017 can be used to retrieve the value of each
 938 symbol on each specific implementation to determine whether the option is supported at
 939 runtime. All headers, data types, and function interfaces required to compile and execute
 940 applications which use the option at runtime (after checking at runtime that the option is
 941 supported) shall be provided, but if the option is not supported at runtime they need not
 942 operate as specified. Utilities or other facilities required only for the option, but not
 943 needed to compile and execute such applications, need not be present.

944 If an option is not supported for compilation, an application that attempts to use anything
 945 associated only with the option is considered to be requiring an extension. Unless explicitly
 946 specified otherwise, the behavior of functions associated with an option that is not supported at
 947 runtime is unspecified, and an application that uses such functions without first checking
 948 `fpathconf()`, `pathconf()`, or `sysconf` is considered to be requiring an extension.

949 Margin codes are defined for each option (see [Section 1.7.1](#), on page 7).

- 950 2.1.6.1 *System Interfaces*
- 951 Refer to [<unistd.h>, Constants for Options and Option Groups](#) (on page 434) for the list of
952 options.
- 953 2.1.6.2 *Shell and Utilities*
- 954 Each of these symbols shall be considered valid names by the implementation. Refer to
955 [<unistd.h>, Constants for Options and Option Groups](#) (on page 434).
- 956 The literal names shown below apply only to the *getconf* utility.
- 957 CD **POSIX2_C_DEV**
- 958 The system supports the C-Language Development Utilities option.
- 959 The utilities in the C-Language Development Utilities option are used for the development
960 of C-language applications, including compilation or translation of C source code and
961 complex program generators for simple lexical tasks and processing of context-free
962 grammars.
- 963 The utilities listed below may be provided by a conforming system; however, any system
964 claiming conformance to the C-Language Development Utilities option shall provide all of
965 the utilities listed.
- 966 *c99*
967 *lex*
968 *yacc*
- 969 **POSIX2_CHAR_TERM**
- 970 The system supports the Terminal Characteristics option. This value need not be present on
971 a system not supporting the User Portability Utilities option.
- 972 Where applicable, the dependency is noted within the description of the utility.
- 973 This option applies only to systems supporting the User Portability Utilities option. If
974 supported, then the system supports at least one terminal type capable of all operations
975 described in POSIX.1-2017; see [Section 10.2](#) (on page 198).
- 976 FD **POSIX2_FORT_DEV**
- 977 The system supports the FORTRAN Development Utilities option.
- 978 The *fort77* FORTRAN compiler is the only utility in the FORTRAN Development Utilities
979 option. This is used for the development of FORTRAN language applications, including
980 compilation or translation of FORTRAN source code.
- 981 The *fort77* utility may be provided by a conforming system; however, any system claiming
982 conformance to the FORTRAN Development Utilities option shall provide the *fort77* utility.
- 983 FR **POSIX2_FORT_RUN**
- 984 The system supports the FORTRAN Runtime Utilities option.
- 985 The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.
- 986 The *asa* utility may be provided by a conforming system; however, any system claiming
987 conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.
- 988 **POSIX2_LOCALEDEF**
- 989 The system supports the Locale Creation Utilities option.
- 990 If supported, the system supports the creation of locales as described in the *localedef* utility.

991 The *localedef* utility may be provided by a conforming system; however, any system
 992 claiming conformance to the Locale Creation Utilities option shall provide the *localedef*
 993 utility.

994 OB BE **POSIX2_PBS**

995 The system supports the Batch Environment Services and Utilities option (see XCU [Chapter](#)
 996 [3](#), on page 2427).

997 **Note:** The Batch Environment Services and Utilities option is a combination of mandatory and
 998 optional batch services and utilities. The POSIX_PBS symbolic constant implies the system
 999 supports all the mandatory batch services and utilities.

1000 POSIX2_PBS_ACCOUNTING

1001 The system supports the Batch Accounting option.

1002 POSIX2_PBS_CHECKPOINT

1003 The system supports the Batch Checkpoint/Restart option.

1004 POSIX2_PBS_LOCATE

1005 The system supports the Locate Batch Job Request option.

1006 POSIX2_PBS_MESSAGE

1007 The system supports the Batch Job Message Request option.

1008 POSIX2_PBS_TRACK

1009 The system supports the Track Batch Job Request option.

1010 SD **POSIX2_SW_DEV**

1011 The system supports the Software Development Utilities option.

1012 The utilities in the Software Development Utilities option are used for the development of
 1013 applications, including compilation or translation of source code, the creation and
 1014 maintenance of library archives, and the maintenance of groups of inter-dependent
 1015 programs.

1016 The utilities listed below may be provided by the conforming system; however, any system
 1017 claiming conformance to the Software Development Utilities option shall provide all of the
 1018 utilities listed here.

1019 *ar*
 1020 *make*
 1021 *nm*
 1022 *strip*

1023 UP **POSIX2_UPE**

1024 The system supports the User Portability Utilities option.

1025 The utilities in the User Portability Utilities option shall be implemented on all systems that
 1026 claim conformance to this option, except for the *vi* utility which is noted as having features
 1027 that cannot be implemented on all terminal types; if the POSIX2_CHAR_TERM option is
 1028 supported, the system shall support all such features on at least one terminal type; see
 1029 [Section 10.2](#) (on page 198).

1030 The list of utilities in the User Portability Utilities option is as follows:

1031 *bg* *fc* *jobs* *talk*
 1032 *ex* *fg* *more* *vi*

1033	XSI	XOPEN_UNIX
1034		The system supports the X/Open System Interfaces (XSI) option (see Section 2.1.4 , on page 19).
1035		
1036	UU	XOPEN_UUCP
1037		The system supports the UUCP Utilities option.
1038		The list of utilities in the UUCP Utilities option is as follows:
1039		<i>uucp</i>
1040		<i>uustat</i>
1041		<i>uux</i>

1042 2.2 Application Conformance

1043 For the purposes of POSIX.1-2017, the application conformance requirements given in this
1044 section apply.

1045 All applications claiming conformance to POSIX.1-2017 shall use only language-dependent
1046 services for the C programming language described in [Section 2.3](#) (on page 31), shall use only
1047 the utilities and facilities defined in the Shell and Utilities volume of POSIX.1-2017, and shall fall
1048 within one of the following categories.

1049 2.2.1 Strictly Conforming POSIX Application

1050 A Strictly Conforming POSIX Application is an application that requires only the facilities
1051 described in POSIX.1-2017. Such an application:

- 1052 1. Shall accept any implementation behavior that results from actions it takes in areas
1053 described in POSIX.1-2017 as *implementation-defined* or *unspecified*, or where POSIX.1-2017
1054 indicates that implementations may vary
- 1055 2. Shall not perform any actions that are described as producing *undefined* results
- 1056 3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-2017,
1057 but shall not rely on any value in the range being greater than the minimums listed or
1058 being less than the maximums listed in POSIX.1-2017
- 1059 4. Shall not use facilities designated as *obsolescent*
- 1060 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1061 facilities whose availability is indicated by [Section 2.1.3](#) (on page 17)
- 1062 6. For the C programming language, shall not produce any output dependent on any
1063 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or
1064 *implementation-defined*, unless the System Interfaces volume of POSIX.1-2017 specifies the
1065 behavior
- 1066 7. For the C programming language, shall not exceed any minimum implementation limit
1067 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of
1068 POSIX.1-2017 specifies a higher minimum implementation limit
- 1069 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200809L before
1070 any header is included

1071 Within POSIX.1-2017, any restrictions placed upon a Conforming POSIX Application shall

1072 restrict a Strictly Conforming POSIX Application.

1073 2.2.2 Conforming POSIX Application

1074 2.2.2.1 ISO/IEC Conforming POSIX Application

1075 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities
1076 described in POSIX.1-2017 and approved Conforming Language bindings for any ISO or IEC
1077 standard. Such an application shall include a statement of conformance that documents all
1078 options and limit dependencies, and all other ISO or IEC standards used.

1079 2.2.2.2 <National Body> Conforming POSIX Application

1080 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming
1081 POSIX Application in that it also may use specific standards of a single ISO/IEC member body
1082 referred to here as <National Body>. Such an application shall include a statement of
1083 conformance that documents all options and limit dependencies, and all other <National Body>
1084 standards used.

1085 2.2.3 Conforming POSIX Application Using Extensions

1086 A Conforming POSIX Application Using Extensions is an application that differs from a
1087 Conforming POSIX Application only in that it uses non-standard facilities that are consistent
1088 with POSIX.1-2017. Such an application shall fully document its requirements for these extended
1089 facilities, in addition to the documentation required of a Conforming POSIX Application. A
1090 Conforming POSIX Application Using Extensions shall be either an ISO/IEC Conforming
1091 POSIX Application Using Extensions or a <National Body> Conforming POSIX Application
1092 Using Extensions (see [Section 2.2.2.1](#) and [Section 2.2.2.2](#)).

1093 2.2.4 Strictly Conforming XSI Application

1094 A Strictly Conforming XSI Application is an application that requires only the facilities
1095 described in POSIX.1-2017. Such an application:

- 1096 1. Shall accept any implementation behavior that results from actions it takes in areas
1097 described in POSIX.1-2017 as *implementation-defined* or *unspecified*, or where POSIX.1-2017
1098 indicates that implementations may vary
- 1099 2. Shall not perform any actions that are described as producing *undefined* results
- 1100 3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-2017,
1101 but shall not rely on any value in the range being greater than the minimums listed or
1102 being less than the maximums listed in POSIX.1-2017
- 1103 4. Shall not use facilities designated as *obsolescent*
- 1104 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
1105 facilities whose availability is indicated by [Section 2.1.4](#) (on page 19)

- 1106 6. For the C programming language, shall not produce any output dependent on any
1107 behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-*
1108 *defined*, unless the System Interfaces volume of POSIX.1-2017 specifies the behavior
- 1109 7. For the C programming language, shall not exceed any minimum implementation limit
1110 defined in the ISO C standard, unless the System Interfaces volume of POSIX.1-2017
1111 specifies a higher minimum implementation limit
- 1112 8. For the C programming language, shall define `_XOPEN_SOURCE` to be 700 before any
1113 header is included
- 1114 Within POSIX.1-2017, any restrictions placed upon a Conforming POSIX Application shall
1115 restrict a Strictly Conforming XSI Application.

1116 **2.2.5 Conforming XSI Application Using Extensions**

1117 A Conforming XSI Application Using Extensions is an application that differs from a Strictly
1118 Conforming XSI Application only in that it uses non-standard facilities that are consistent with
1119 POSIX.1-2017. Such an application shall fully document its requirements for these extended
1120 facilities, in addition to the documentation required of a Strictly Conforming XSI Application.

1121 **2.3 Language-Dependent Services for the C Programming Language**

1122 Implementors seeking to claim conformance using the ISO C standard shall claim POSIX
1123 conformance as described in [Section 2.1.3](#) (on page 17).

1124 **2.4 Other Language-Related Specifications**

1125 POSIX.1-2017 is currently specified in terms of the shell command language and ISO C. Bindings
1126 to other programming languages are being developed.

1127 If conformance to POSIX.1-2017 is claimed for implementation of any programming language,
1128 the implementation of that language shall support the use of external symbols distinct to at least
1129 31 bytes in length in the source program text. (That is, identifiers that differ at or before the
1130 thirty-first byte shall be distinct.) If a national or international standard governing a language
1131 defines a maximum length that is less than this value, the language-defined maximum shall be
1132 supported. External symbols that differ only by case shall be distinct when the character set in
1133 use distinguishes uppercase and lowercase characters and the language permits (or requires)
1134 uppercase and lowercase characters to be distinct in external symbols.

1135

Chapter 3

1136

Definitions

1137 For the purposes of POSIX.1-2017, the following terms and definitions apply. The Authoritative
1138 Dictionary of IEEE Standards Terms, Seventh Edition should be referenced for terms not defined
1139 in this section.

1140 **Note:** No shading to denote extensions or options occurs in this chapter. Where the terms and
1141 definitions given in this chapter are used elsewhere in text related to extensions and options,
1142 they are shaded as appropriate.

1143 3.1 Abortive Release

1144 An abrupt termination of a network connection that may result in the loss of data.

1145 3.2 Absolute Pathname

1146 A pathname beginning with a single or more than two <slash> characters; see also [Section 3.271](#)
1147 (on page 76).

1148 **Note:** Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

1149 3.3 Access Mode

1150 A particular form of access permitted to a file.

1151 3.4 Additional File Access Control Mechanism

1152 An implementation-defined mechanism that is layered upon the access control mechanisms
1153 defined here, but which do not grant permissions beyond those defined herein, although they
1154 may further restrict them.

1155 **Note:** File Access Permissions are defined in detail in [Section 4.5](#) (on page 108).

1156 3.5 Address Space

1157 The memory locations that can be referenced by a process or the threads of a process.

1158 3.6 Advisory Information

1159 An interface that advises the implementation on (portable) application behavior so that it can
1160 optimize the system.

1161 3.7 Affirmative Response

1162 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
1163 keyword **yesexpr**, matching an extended regular expression in the current locale.

1164 **Note:** The *LC_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 165).

1165 3.8 Alert

1166 To cause the user's terminal to give some audible or visual indication that an error or some other
1167 event has occurred. When the standard output is directed to a terminal device, the method for
1168 alerting the terminal user is unspecified. When the standard output is not directed to a terminal
1169 device, the alert is accomplished by writing the alert to standard output (unless the utility
1170 description indicates that the use of standard output produces undefined results in this case).

1171 3.9 Alert Character (<alert>)

1172 A character that in the output stream should cause a terminal to alert its user via a visual or
1173 audible notification. It is the character designated by '`\a`' in the C language. It is unspecified
1174 whether this character is the exact sequence transmitted to an output device by the system to
1175 accomplish the alert function.

1176 3.10 Alias Name

1177 In the shell command language, a word consisting solely of underscores, digits, and alphabetic
1178 characters from the portable character set and any of the following characters: '`!`', '`%`', '`'`', '`,`', '`@`'.

1179 Implementations may allow other characters within alias names as an extension.

1180 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

1181 3.11 Alignment

1182 A requirement that objects of a particular type be located on storage boundaries with addresses
1183 that are particular multiples of a byte address.

1184 **Note:** See also the ISO C standard, Section B3.

1185 **3.12 Alternate File Access Control Mechanism**

1186 An implementation-defined mechanism that is independent of the access control mechanisms
1187 defined herein, and which if enabled on a file may either restrict or extend the permissions of a
1188 given user. POSIX.1-2017 defines when such mechanisms can be enabled and when they are
1189 disabled.

1190 **Note:** File Access Permissions are defined in detail in [Section 4.5](#) (on page 108).

1191 **3.13 Alternate Signal Stack**

1192 Memory associated with a thread, established upon request by the implementation for a thread,
1193 separate from the thread signal stack, in which signal handlers responding to signals sent to that
1194 thread may be executed.

1195 **3.14 Ancillary Data**

1196 Protocol-specific, local system-specific, or optional information. The information can be both
1197 local or end-to-end significant, header information, part of a data portion, protocol-specific, and
1198 implementation or system-specific.

1199 **3.15 Angle Brackets**

1200 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase
1201 ``enclosed in angle brackets'', the symbol '`<`' immediately precedes the object to be enclosed,
1202 and '`>`' immediately follows it. When describing these characters in the portable character set,
1203 the names `<less-than-sign>` and `<greater-than-sign>` are used.

1204 **3.16 Apostrophe Character (<apostrophe>)**

1205 The character designated by '`'`' in the C language, also known as the single-quote character.

1206 3.17 Application

1207 A computer program that performs some desired function.

1208 When the User Portability Utilities option is supported, requirements placed on applications
1209 relating to the use of standard utilities shall also apply to the actions of a user who is entering
1210 shell command language statements into an interactive shell.

1211 3.18 Application Address

1212 Endpoint address of a specific application.

1213 3.19 Application Program Interface (API)

1214 The definition of syntax and semantics for providing computer system services.

1215 3.20 Appropriate Privileges

1216 An implementation-defined means of associating privileges with a process with regard to the
1217 function calls, function call options, and the commands that need special privileges. There may
1218 be zero or more such means. These means (or lack thereof) are described in the conformance
1219 document.

1220 **Note:** Function calls are defined in the System Interfaces volume of POSIX.1-2017, and commands are
1221 defined in the Shell and Utilities volume of POSIX.1-2017.

1222 3.21 Argument

1223 In the shell command language, a parameter passed to a utility as the equivalent of a single
1224 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,
1225 option-arguments, or operands following the command name.

1226 **Note:** The Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213) and XCU [Section](#)
1227 [2.9.1.1](#) (on page 2367).

1228 In the C language, an expression in a function call expression or a sequence of preprocessing
1229 tokens in a function-like macro invocation.

1230 3.22 Arm (a Timer)

1231 To start a timer measuring the passage of time, enabling notifying a process when the specified
1232 time or time interval has passed.

1233 3.23 Asterisk Character (<asterisk>)

1234 The character ' * '.

1235 3.24 Async-Cancel-Safe Function

1236 A function that may be safely invoked by an application while the asynchronous form of
1237 cancellation is enabled. No function is async-cancel-safe unless explicitly described as such.

1238 3.25 Asynchronous Events

1239 Events that occur independently of the execution of the application.

1240 3.26 Asynchronous Input and Output

1241 A functionality enhancement to allow an application process to queue data input and output
1242 commands with asynchronous notification of completion.

1243 3.27 Async-Signal-Safe Function

1244 A function that can be called, without restriction, from signal-catching functions. Note that,
1245 although there is no restriction on the calls themselves, for certain functions there are restrictions
1246 on subsequent behavior after the function is called from a signal-catching function. No function
1247 is async-signal-safe unless explicitly described as such.

1248 **Note:** Async-signal-safety is defined in detail in XSH [Section 2.4.3](#) (on page 490).

1249 3.28 Asynchronously-Generated Signal

1250 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals
1251 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a
1252 property of how the signal was generated and not a property of the signal number. All signals
1253 may be generated asynchronously.

1254 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of POSIX.1-2017.

1255 3.29 Asynchronous I/O Completion

1256 For an asynchronous read or write operation, when a corresponding synchronous read or write
1257 would have completed and when any associated status fields have been updated.

1258 3.30 Asynchronous I/O Operation

1259 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from
1260 further use of the processor.

1261 This implies that the process and the I/O operation may be running concurrently.

1262 3.31 Authentication

1263 The process of validating a user or process to verify that the user or process is not a counterfeit.

1264 3.32 Authorization

1265 The process of verifying that a user or process has permission to use a resource in the manner
1266 requested.

1267 To ensure security, the user or process would also need to be authenticated before granting
1268 access.

1269 3.33 Background Job

1270 See *Background Process Group* in [Section 3.35](#).

1271 3.34 Background Process

1272 A process that is a member of a background process group.

1273 3.35 Background Process Group (or Background Job)

1274 Any process group, other than a foreground process group, that is a member of a session that
1275 has established a connection with a controlling terminal.

1276 3.36 Backquote Character

1277 The character '`', also known as <grave-accent>.

1278 3.37 Backslash Character (<backslash>)

1279 The character designated by '\\\' in the C language, also known as reverse solidus.

1280 **3.38 Backspace Character (<backspace>)**

1281 A character that, in the output stream, should cause printing (or displaying) to occur one
1282 column position previous to the position about to be printed. If the position about to be printed
1283 is at the beginning of the current line, the behavior is unspecified. It is the character designated
1284 by '\b' in the C language. It is unspecified whether this character is the exact sequence
1285 transmitted to an output device by the system to accomplish the backspace function. The
1286 backspace defined here is not necessarily the ERASE special character.

1287 **Note:** Special Characters are defined in detail in [Section 11.1.9](#) (on page 204).

1288 **3.39 Barrier**

1289 A synchronization object that allows multiple threads to synchronize at a particular point in
1290 their execution.

1291 **3.40 Basename**

1292 For pathnames containing at least one filename: the final, or only, filename in the pathname. For
1293 pathnames consisting only of <slash> characters: either '/' or '/' if the pathname consists of
1294 exactly two <slash> characters, and '/' otherwise.

1295 **3.41 Basic Regular Expression (BRE)**

1296 A regular expression (see [Section 3.321](#), on page 85) used by the majority of utilities that select
1297 strings from a set of character strings.

1298 **Note:** Basic Regular Expressions are described in detail in [Section 9.3](#) (on page 183).

1299 **3.42 Batch Access List**

1300 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a
1301 batch queue.

1302 A batch access list is associated with a batch queue. A batch server uses the batch access list of a
1303 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1304 **3.43 Batch Administrator**

1305 A user that is authorized to modify all the attributes of queues and jobs and to change the status
1306 of a batch server.

1307 **3.44 Batch Client**

1308 A computational entity that utilizes batch services by making requests of batch servers.

1309 Batch clients often provide the means by which users access batch services, although a batch
1310 server may act as a batch client by virtue of making requests of another batch server.

1311 **3.45 Batch Destination**

1312 The batch server in a batch system to which a batch job should be sent for processing.

1313 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.
1314 A batch destination may consist of a batch server-specific portion, a network-wide portion, or
1315 both. The batch server-specific portion is referred to as the “batch queue”. The network-wide
1316 portion is referred to as a “batch server name”.

1317 **3.46 Batch Destination Identifier**

1318 A string that identifies a specific batch destination.

1319 A string of characters in the portable character set used to specify a particular batch destination.

1320 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

1321 **3.47 Batch Directive**

1322 A line from a file that is interpreted by the batch server. The line is usually in the form of a
1323 comment and is an additional means of passing options to the *qsub* utility.

1324 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2017.

1325 **3.48 Batch Job**

1326 A set of computational tasks for a computing system.

1327 Batch jobs are managed by batch servers.

1328 Once created, a batch job may be executing or pending execution. A batch job that is executing
1329 has an associated session leader (a process) that initiates and monitors the computational tasks
1330 of the batch job.

1331 3.49 Batch Job Attribute

1332 A named data type whose value affects the processing of a batch job.

1333 The values of the attributes of a batch job affect the processing of that job by the batch server that
1334 manages the batch job.

1335 3.50 Batch Job Identifier

1336 A unique name for a batch job. A name that is unique among all other batch job identifiers in a
1337 batch system and that identifies the batch server to which the batch job was originally
1338 submitted.

1339 3.51 Batch Job Name

1340 A label that is an attribute of a batch job. The batch job name is not necessarily unique.

1341 3.52 Batch Job Owner

1342 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see
1343 also [Section 3.437](#), on page 103) and *hostname* is a network host name.

1344 3.53 Batch Job Priority

1345 A value specified by the user that may be used by an implementation to determine the order in
1346 which batch jobs are selected to be executed. Job priority has a numeric value in the range
1347 -1 024 to 1 023.

1348 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.

1349 3.54 Batch Job State

1350 An attribute of a batch job which determines the types of requests that the batch server that
1351 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING,
1352 HELD, WAITING, EXITING, and TRANSITING.

1353 3.55 Batch Name Service

1354 A service that assigns batch names that are unique within the batch name space, and that can
1355 translate a unique batch name into the location of the named batch entity.

1356 **3.56 Batch Name Space**

1357 The environment within which a batch name is known to be unique.

1358 **3.57 Batch Node**

1359 A host containing part or all of a batch system.

1360 A batch node is a host meeting at least one of the following conditions:

1361 Capable of executing a batch client

1362 Contains a routing batch queue

1363 Contains an execution batch queue

1364 **3.58 Batch Operator**

1365 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and
1366 may change the status of the batch server.

1367 **3.59 Batch Queue**

1368 A manageable object that represents a set of batch jobs and is managed by a single batch server.

1369 **Note:** A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from
1370 the batch queue for execution based on attributes such as priority, resource requirements, and
1371 hold conditions.

1372 See also XCU [Section 3.1.2](#) (on page 2428).

1373 **3.60 Batch Queue Attribute**

1374 A named data type whose value affects the processing of all batch jobs that are members of the
1375 batch queue.

1376 A batch queue has attributes that affect the processing of batch jobs that are members of the
1377 batch queue.

1378 **3.61 Batch Queue Position**

1379 The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue.
1380 This is defined in part by submission time and priority; see also [Section 3.62](#) (on page 43).

1381 **3.62 Batch Queue Priority**

1382 The maximum job priority allowed for any batch job in a given batch queue.

1383 The batch queue priority is set and may be changed by users with appropriate privileges. The
1384 priority is bounded in an implementation-defined manner.

1385 **3.63 Batch Rerunability**

1386 An attribute of a batch job indicating that it may be rerun after an abnormal termination from
1387 the beginning without affecting the validity of the results.

1388 **3.64 Batch Restart**

1389 The action of resuming the processing of a batch job from the point of the last checkpoint.
1390 Typically, this is done if the batch job has been interrupted because of a system failure.

1391 **3.65 Batch Server**

1392 A computational entity that provides batch services.

1393 **3.66 Batch Server Name**

1394 A string of characters in the portable character set used to specify a particular server in a
1395 network.

1396 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

1397 **3.67 Batch Service**

1398 Computational and organizational services performed by a batch system on behalf of batch jobs.

1399 Batch services are of two types: requested and deferred.

1400 **Note:** Batch Services are listed in XCU [Table 3-5](#) (on page 2442).

1401 **3.68 Batch Service Request**

1402 A solicitation of services from a batch client to a batch server.

1403 A batch service request may entail the exchange of any number of messages between the batch
1404 client and the batch server.

1405 When naming specific types of service requests, the term “request” is qualified by the type of
1406 request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

1407 **3.69 Batch Submission**

1408 The process by which a batch client requests that a batch server create a batch job via a *Queue Job*
1409 *Request* to perform a specified computational task.

1410 **3.70 Batch System**

1411 A collection of one or more batch servers.

1412 **3.71 Batch Target User**

1413 The name of a user on the batch destination batch server.

1414 The target user is the user name under whose account the batch job is to execute on the
1415 destination batch server.

1416 **3.72 Batch User**

1417 A user who is authorized to make use of batch services.

1418 **3.73 Bind**

1419 The process of assigning a network address to an endpoint.

1420 **3.74 Blank Character (<blank>)**

1421 One of the characters that belong to the **blank** character class as defined via the *LC_CTYPE*
1422 category in the current locale. In the POSIX locale, a <blank> character is either a <tab> or a
1423 <space>.

1424 3.75 Blank Line

1425 A line consisting solely of zero or more <blank> characters terminated by a <newline>; see also
1426 [Section 3.145](#) (on page 56).

1427 3.76 Blocked Process (or Thread)

1428 A process (or thread) that is waiting for some condition (other than the availability of a
1429 processor) to be satisfied before it can continue execution.

1430 3.77 Blocking

1431 A property of an open file description that causes function calls associated with it to wait for the
1432 requested action to be performed before returning.

1433 3.78 Block-Mode Terminal

1434 A terminal device operating in a mode incapable of the character-at-a-time input and output
1435 operations described by some of the standard utilities.

1436 **Note:** Output Devices and Terminal Types are defined in detail in [Section 10.2](#) (on page 198).

1437 3.79 Block Special File

1438 A file that refers to a device. A block special file is normally distinguished from a character
1439 special file by providing access to the device in a manner such that the hardware characteristics
1440 of the device are not visible.

1441 3.80 Braces

1442 The characters '{' (left-curly-bracket) and '}' (right-curly-bracket). When used in the phrase
1443 "enclosed in (curly) braces" the symbol '{' immediately precedes the object to be enclosed, and
1444 '}' immediately follows it. When describing these characters in the portable character set, the
1445 names <left-curly-bracket> and <left-brace> are used for '{', and <right-curly-bracket> and
1446 <right-brace> are used for '}'.

1447 3.81 Brackets

1448 The characters '[' (left-square-bracket) and ']' (right-square-bracket). When used in the
1449 phrase "enclosed in (square) brackets" the symbol '[' immediately precedes the object to be
1450 enclosed, and ']' immediately follows it. When describing these characters in the portable
1451 character set, the names <left-square-bracket> and <right-square-bracket> are used.

1452 3.82 Broadcast

1453 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and
1454 RFC 922.

1455 3.83 Built-In Utility (or Built-In)

1456 A utility implemented within a shell. The utilities referred to as special built-ins have special
1457 qualities. Unless qualified, the term “built-in” includes the special built-in utilities. Regular
1458 built-ins are not required to be actually built into the shell on the implementation, but they do
1459 have special command-search qualities.

1460 **Note:** Special Built-In Utilities are defined in detail in XCU [Section 2.14](#) (on page 2384).

1461 Regular Built-In Utilities are defined in detail in XCU [Section 2.9.1.1](#) (on page 2367).

1462 3.84 Byte

1463 An individually addressable unit of data storage that is exactly an octet, used to store a character
1464 or a portion of a character; see also [Section 3.87](#) (on page 47). A byte is composed of a
1465 contiguous sequence of 8 bits. The least significant bit is called the “low-order” bit; the most
1466 significant is called the “high-order” bit.

1467 **Note:** The definition of byte from the ISO C standard is broader than the above and might
1468 accommodate hardware architectures with different sized addressable units than octets.

1469 3.85 Byte Input/Output Functions

1470 The functions that perform byte-oriented input from streams or byte-oriented output to streams:
1471 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *getdelim()*,
1472 *getline()*, *gets()*, *printf()*, *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1473 **Note:** Functions are defined in detail in the System Interfaces volume of POSIX.1-2017.

1474 3.86 Carriage-Return Character (<carriage-return>)

1475 A character that in the output stream indicates that printing should start at the beginning of the
1476 same physical line in which the carriage-return occurred. It is the character designated by ‘\r’
1477 in the C language. It is unspecified whether this character is the exact sequence transmitted to an
1478 output device by the system to accomplish the movement to the beginning of the line.

1479 3.87 Character

1480 A sequence of one or more bytes representing a single graphic symbol or control code.

1481 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte
1482 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,
1483 *character* here has no necessary relationship with storage space, and *byte* is used when storage
1484 space is discussed.

1485 See the definition of the portable character set in [Section 6.1](#) (on page 125) for a further
1486 explanation of the graphical representations of (abstract) characters, as opposed to character
1487 encodings.

1488 3.88 Character Array

1489 An array of elements of type **char**.

1490 3.89 Character Class

1491 A named set of characters sharing an attribute associated with the name of the class. The classes
1492 and the characters that they contain are dependent on the value of the *LC_CTYPE* category in
1493 the current locale.

1494 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

1495 3.90 Character Set

1496 A finite set of different characters used for the representation, organization, or control of data.

1497 3.91 Character Special File

1498 A file that refers to a device (such as a terminal device file) or that has special properties (such as
1499 */dev/null*).

1500 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

1501 3.92 Character String

1502 A contiguous sequence of characters terminated by and including the first null byte.

1503 3.93 Child Process

1504 A new process created (by *fork()*, *posix_spawn()*, or *posix_spawnp()*) by a given process. A child
1505 process remains the child of the creating process as long as both processes continue to exist.

1506 **Note:** The *fork()*, *posix_spawn()*, and *posix_spawnp()* functions are defined in detail in the System
1507 Interfaces volume of POSIX.1-2017.

1508 3.94 Circumflex Character (<circumflex>)

1509 The character ' ^ '.

1510 3.95 Clock

1511 A software or hardware object that can be used to measure the apparent or actual passage of
1512 time.

1513 The current value of the time measured by a clock can be queried and, possibly, set to a value
1514 within the legal range of the clock.

1515 3.96 Clock Jump

1516 The difference between two successive distinct values of a clock, as observed from the
1517 application via one of the "get time" operations.

1518 3.97 Clock Tick

1519 An interval of time; an implementation-defined number of these occur each second. Clock ticks
1520 are one of the units that may be used to express a value found in type `clock_t`.

1521 3.98 Coded Character Set

1522 A set of unambiguous rules that establishes a character set and the one-to-one relationship
1523 between each character of the set and its bit representation.

1524 3.99 Codeset

1525 The result of applying rules that map a numeric code value to each element of a character set.
1526 An element of a character set may be related to more than one numeric code value but the
1527 reverse is not true. However, for state-dependent encodings the relationship between numeric
1528 code values and elements of a character set may be further controlled by state information. The
1529 character set may contain fewer elements than the total number of possible numeric code values;
1530 that is, some code values may be unassigned.

1531 **Note:** Character Encoding is defined in detail in [Section 6.2](#) (on page 128).

1532 3.100 Collating Element

1533 The smallest entity used to determine the logical ordering of character or wide-character strings;
1534 see also [Section 3.102](#). A collating element consists of either a single character, or two or more
1535 characters collating as a single entity. The value of the *LC_COLLATE* category in the current
1536 locale determines the current set of collating elements.

1537 3.101 Collation

1538 The logical ordering of character or wide-character strings according to defined precedence
1539 rules. These rules identify a collation sequence between the collating elements, and such
1540 additional rules that can be used to order strings consisting of multiple collating elements.

1541 3.102 Collation Sequence

1542 The relative order of collating elements as determined by the setting of the *LC_COLLATE*
1543 category in the current locale. The collation sequence is used for sorting and is determined from
1544 the collating weights assigned to each collating element. In the absence of weights, the collation
1545 sequence is the order in which collating elements are specified between **order_start** and
1546 **order_end** keywords in the *LC_COLLATE* category.

1547 Multi-level sorting is accomplished by assigning elements one or more collation weights, up to
1548 the limit {*COLL_WEIGHTS_MAX*}. On each level, elements may be given the same weight (at
1549 the primary level, called an equivalence class; see also [Section 3.151](#), on page 57) or be omitted
1550 from the sequence. Strings that collate equally using the first assigned weight (primary ordering)
1551 are then compared using the next assigned weight (secondary ordering), and so on.

1552 **Note:** {*COLL_WEIGHTS_MAX*} is defined in detail in [<limits.h>](#).

1553 3.103 Column Position

1554 A unit of horizontal measure related to characters in a line.

1555 It is assumed that each character in a character set has an intrinsic column width independent of
1556 any output device. Each printable character in the portable character set has a column width of
1557 one. The standard utilities, when used as described in POSIX.1-2017, assume that all characters
1558 have integral column widths. The column width of a character is not necessarily related to the
1559 internal representation of the character (numbers of bits or bytes).

1560 The column position of a character in a line is defined as one plus the sum of the column widths
1561 of the preceding characters in the line. Column positions are numbered starting from 1.

1562 3.104 Command

1563 A directive to the shell to perform a particular task.

1564 **Note:** Shell Commands are defined in detail in XCU [Section 2.9](#) (on page 2365).

1565 3.105 Command Language Interpreter

1566 An interface that interprets sequences of text input as commands. It may operate on an input
1567 stream or it may interactively prompt and read commands from a terminal. It is possible for
1568 applications to invoke utilities through a number of interfaces, which are collectively considered
1569 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*
1570 function, although *popen()* and the various forms of *exec* may also be considered to behave as
1571 interpreters.

1572 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2017.

1573 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume
1574 of POSIX.1-2017.

1575 3.106 Composite Graphic Symbol

1576 A graphic symbol consisting of a combination of two or more other graphic symbols in a single
1577 character position, such as a diacritical mark and a base character.

1578 3.107 Condition Variable

1579 A synchronization object which allows a thread to suspend execution, repeatedly, until some
1580 associated predicate becomes true. A thread whose execution is suspended on a condition
1581 variable is said to be blocked on the condition variable.

1582 **3.108 Connected Socket**

1583 A connection-mode socket for which a connection has been established, or a connectionless-
 1584 mode socket for which a peer address has been set. See also [Section 3.109](#), [Section 3.110](#), [Section](#)
 1585 [3.111](#), and [Section 3.356](#) (on page 91).

1586 **3.109 Connection**

1587 An association established between two or more endpoints for the transfer of data

1588 **3.110 Connection Mode**

1589 The transfer of data in the context of a connection; see also [Section 3.111](#).

1590 **3.111 Connectionless Mode**

1591 The transfer of data other than in the context of a connection; see also [Section 3.110](#) and [Section](#)
 1592 [3.124](#) (on page 53).

1593 **3.112 Control Character**

1594 A character, other than a graphic character, that affects the recording, processing, transmission,
 1595 or interpretation of text.

1596 **3.113 Control Operator**

1597 In the shell command language, a token that performs a control function. It is one of the
 1598 following symbols:

1599 `& && () ; ;; newline | ||`

1600 The end-of-input indicator used internally by the shell is also considered a control operator.

1601 **Note:** Token Recognition is defined in detail in XCU [Section 2.3](#) (on page 2347).

1602 **3.114 Controlling Process**

1603 The session leader that established the connection to the controlling terminal. If the terminal
 1604 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be
 1605 the controlling process.

1606 **3.115 Controlling Terminal**

1607 A terminal that is associated with a session. Each session may have at most one controlling
1608 terminal associated with it, and a controlling terminal is associated with exactly one session.
1609 Certain input sequences from the controlling terminal cause signals to be sent to all processes in
1610 the foreground process group associated with the controlling terminal.

1611 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

1612 **3.116 Conversion Descriptor**

1613 A per-process unique value used to identify an open codeset conversion.

1614 **3.117 Core File**

1615 A file of unspecified format that may be generated when a process terminates abnormally.

1616 **3.118 CPU Time (Execution Time)**

1617 The time spent executing a process or thread, including the time spent executing system services
1618 on behalf of that process or thread. If the Threads option is supported, then the value of the
1619 CPU-time clock for a process is implementation-defined. With this definition the sum of all the
1620 execution times of all the threads in a process might not equal the process execution time, even
1621 in a single-threaded process, because implementations may differ in how they account for time
1622 during context switches or for other reasons.

1623 **3.119 CPU-Time Clock**

1624 A clock that measures the execution time of a particular process or thread.

1625 **3.120 CPU-Time Timer**

1626 A timer attached to a CPU-time clock.

1627 **3.121 Current Job**

1628 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There
1629 is at most one current job; see also [Section 3.204](#) (on page 66).

1630 **3.122 Current Working Directory**

1631 See *Working Directory* in [Section 3.447](#) (on page 105).

1632 **3.123 Cursor Position**

1633 The line and column position on the screen denoted by the terminal's cursor.

1634 **3.124 Datagram**

1635 A unit of data transferred from one endpoint to another in connectionless mode service.

1636 **3.125 Data Segment**

1637 Memory associated with a process, that can contain dynamically allocated data.

1638 **3.126 Deferred Batch Service**

1639 A service that is performed as a result of events that are asynchronous with respect to requests.

1640 **Note:** Once a batch job has been created, it is subject to deferred services.

1641 **3.127 Device**

1642 A computer peripheral or an object that appears to the application as such.

1643 **3.128 Device ID**

1644 A non-negative integer used to identify a device.

1645 **3.129 Directory**

1646 A file that contains directory entries. No two directory entries in the same directory have the
1647 same name.

1648 **3.130 Directory Entry (or Link)**

1649 An object that associates a filename with a file. Several directory entries can associate names

1650 with the same file.

1651 **3.131 Directory Stream**

1652 A sequence of all the directory entries in a particular directory. An open directory stream may be
1653 implemented using a file descriptor.

1654 **3.132 Disarm (a Timer)**

1655 To stop a timer from measuring the passage of time, disabling any future process notifications
1656 (until the timer is armed again).

1657 **3.133 Display**

1658 To output to the user's terminal. If the output is not directed to a terminal, the results are
1659 undefined.

1660 **3.134 Display Line**

1661 A line of text on a physical device or an emulation thereof. Such a line will have a maximum
1662 number of characters which can be presented.

1663 **Note:** This may also be written as "line on the display".

1664 **3.135 Dollar-Sign Character (<dollar-sign>)**

1665 The character '\$'.

1666 **3.136 Dot**

1667 In the context of naming files, the filename consisting of a single <period> character ('.').

1668 **Note:** In the context of shell special built-in utilities, see *dot* in XCU [Section 2.14](#) (on page 2384).

1669 Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

1670 3.137 Dot-Dot

1671 The filename consisting solely of two <period> characters (" . . ").

1672 **Note:** Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

1673 3.138 Double-Quote Character

1674 The character ' " ', also known as <quotation-mark>.

1675 **Note:** The “double” adjective in this term refers to the two strokes in the character glyph.
1676 POSIX.1-2017 never uses the term “double-quote” to refer to two apostrophes or quotation-
1677 marks.

1678 3.139 Downshifting

1679 The conversion of an uppercase character that has a single-character lowercase representation
1680 into this lowercase representation.

1681 3.140 Driver

1682 A module that controls data transferred to and received from devices.

1683 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are
1684 frequently written separately from the writing of the implementation. A driver may contain
1685 processor-specific code, and therefore be non-portable.

1686 3.141 Effective Group ID

1687 An attribute of a process that is used in determining various permissions, including file access
1688 permissions; see also [Section 3.189](#) (on page 63).

1689 3.142 Effective User ID

1690 An attribute of a process that is used in determining various permissions, including file access
1691 permissions; see also [Section 3.436](#) (on page 103).

1692 3.143 Eight-Bit Transparency

1693 The ability of a software component to process 8-bit characters without modifying or utilizing
1694 any part of the character in a way that is inconsistent with the rules of the current coded
1695 character set.

1696 **3.144 Empty Directory**

1697 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link
1698 to it (other than its own dot entry, if one exists), in dot-dot. No other links to the directory may
1699 exist. It is unspecified whether an implementation can ever consider the root directory to be
1700 empty.

1701 **3.145 Empty Line**

1702 A line consisting of only a <newline>; see also [Section 3.75](#) (on page 45).

1703 **3.146 Empty String (or Null String)**

1704 A string whose first byte is a null byte.

1705 **3.147 Empty Wide-Character String**

1706 A wide-character string whose first element is a null wide-character code.

1707 **3.148 Encoding Rule**

1708 The rules used to convert between wide-character codes and multi-byte character codes.

1709 **Note:** Stream Orientation and Encoding Rules are defined in detail in XSH [Section 2.5.2](#) (on page 498).

1710 **3.149 Entire Regular Expression**

1711 The concatenated set of one or more basic regular expressions or extended regular expressions
1712 that make up the pattern specified for string selection.

1713 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 181).

1714 3.150 Epoch

1715 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time
1716 (UTC).

1717 **Note:** See also *Seconds Since the Epoch* defined in [Section 4.16](#) (on page 113).

1718 3.151 Equivalence Class

1719 A set of collating elements with the same primary collation weight.

1720 Elements in an equivalence class are typically elements that naturally group together, such as all
1721 accented letters based on the same base letter.

1722 The collation order of elements within an equivalence class is determined by the weights
1723 assigned on any subsequent levels after the primary weight.

1724 3.152 Era

1725 A locale-specific method for counting and displaying years.

1726 **Note:** The *LC_TIME* category is defined in detail in [Section 7.3.5](#) (on page 159).

1727 3.153 Event Management

1728 The mechanism that enables applications to register for and be made aware of external events
1729 such as data becoming available for reading.

1730 3.154 Executable File

1731 A regular file acceptable as a new process image file by the equivalent of the *exec* family of
1732 functions, and thus usable as one form of a utility. The standard utilities described as compilers
1733 can produce executable files, but other unspecified methods of producing executable files may
1734 also be provided. The internal format of an executable file is unspecified, but a conforming
1735 application cannot assume an executable file is a text file.

1736 3.155 Execute

1737 To perform command search and execution actions, as defined in the Shell and Utilities volume
1738 of POSIX.1-2017; see also [Section 3.201](#) (on page 65).

1739 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.1](#) (on page 2367).

1740 3.156 Execution Time

1741 See *CPU Time* in [Section 3.118](#) (on page 52).

1742 3.157 Execution Time Monitoring

1743 A set of execution time monitoring primitives that allow online measuring of thread and process
1744 execution times.

1745 3.158 Expand

1746 In the shell command language, when not qualified, the act of applying word expansions.

1747 **Note:** Word Expansions are defined in detail in XCU [Section 2.6](#) (on page 2353).

1748 3.159 Extended Regular Expression (ERE)

1749 A regular expression (see also [Section 3.321](#), on page 85) that is an alternative to the Basic
1750 Regular Expression using a more extensive syntax, occasionally used by some utilities.

1751 **Note:** Extended Regular Expressions are described in detail in [Section 9.4](#) (on page 188).

1752 3.160 Extended Security Controls

1753 Implementation-defined security controls allowed by the file access permission and appropriate
1754 privileges (see also [Section 3.20](#), on page 36) mechanisms, through which an implementation can
1755 support different security policies from those described in POSIX.1-2017.

1756 **Note:** See also *Extended Security Controls* defined in [Section 4.4](#) (on page 108).

1757 File Access Permissions are defined in detail in [Section 4.5](#) (on page 108).

1758 **3.161 Feature Test Macro**

1759 A macro used to determine whether a particular set of features is included from a header.

1760 **Note:** See also XSH [Section 2.2](#) (on page 472).

1761 **3.162 Field**

1762 In the shell command language, a unit of text that is the result of parameter expansion,
1763 arithmetic expansion, command substitution, or field splitting. During command processing, the
1764 resulting fields are used as the command name and its arguments.

1765 **Note:** Parameter Expansion is defined in detail in XCU [Section 2.6.2](#) (on page 2354).

1766 Arithmetic Expansion is defined in detail in XCU [Section 2.6.4](#) (on page 2358).

1767 Command Substitution is defined in detail in XCU [Section 2.6.3](#) (on page 2357).

1768 Field Splitting is defined in detail in XCU [Section 2.6.5](#) (on page 2359).

1769 For further information on command processing, see XCU [Section 2.9.1](#) (on page 2365).

1770 **3.163 FIFO Special File (or FIFO)**

1771 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

1772 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of POSIX.1-2017,
1773 *lseek()*, *open()*, *read()*, and *write()*.

1774 **3.164 File**

1775 An object that can be written to, or read from, or both. A file has certain attributes, including
1776 access permissions and type. File types include regular file, character special file, block special
1777 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported
1778 by the implementation.

1779 **3.165 File Description**

1780 See *Open File Description* in [Section 3.258](#) (on page 74).

1781 3.166 File Descriptor

1782 A per-process unique, non-negative integer used to identify an open file for the purpose of file
1783 access. The value of a newly-created file descriptor is from zero to {OPEN_MAX}-1. A file
1784 descriptor can have a value greater than or equal to {OPEN_MAX} if the value of {OPEN_MAX}
1785 has decreased (see *sysconf()*) since the file descriptor was opened. File descriptors may also be
1786 used to implement message catalog descriptors and directory streams; see also [Section 3.258](#) (on
1787 page 74).

1788 **Note:** {OPEN_MAX} is defined in detail in [<limits.h>](#).

1789 3.167 File Group Class

1790 The property of a file indicating access permissions for a process related to the group
1791 identification of a process. A process is in the file group class of a file if the process is not in the
1792 file owner class and if the effective group ID or one of the supplementary group IDs of the
1793 process matches the group ID associated with the file. Other members of the class may be
1794 implementation-defined.

1795 3.168 File Mode

1796 An object containing the file mode bits and some information about the file type of a file.

1797 **Note:** File mode bits and file types are defined in detail in [<sys/stat.h>](#).

1798 3.169 File Mode Bits

1799 A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), set-group-ID-on-execution
1800 bit (S_ISGID), and, on directories, the restricted deletion flag bit (S_ISVTX).

1801 **Note:** File Mode Bits are defined in detail in [<sys/stat.h>](#).

1802 3.170 Filename

1803 A sequence of bytes consisting of 1 to {NAME_MAX} bytes used to name a file. The bytes
1804 composing the name shall not contain the <NUL> or <slash> characters. In the context of a
1805 pathname, each filename shall be followed by a <slash> or a <NUL> character; elsewhere, a
1806 filename followed by a <NUL> character forms a string (but not necessarily a character string).
1807 The filenames **dot** and **dot-dot** have special meaning. A filename is sometimes referred to as a
1808 "pathname component". See also [Section 3.271](#) (on page 76).

1809 **Note:** Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

1810 3.171 Filename String

1811 A string consisting of a filename followed by a <NUL> character.

1812 3.172 File Offset

1813 The byte position in the file where the next I/O operation begins. Each open file description
1814 associated with a regular file, block special file, or directory has a file offset. A character special
1815 file that does not refer to a terminal device may have a file offset. There is no file offset specified
1816 for a pipe or FIFO.

1817 3.173 File Other Class

1818 The property of a file indicating access permissions for a process related to the user and group
1819 identification of a process. A process is in the file other class of a file if the process is not in the
1820 file owner class or file group class.

1821 3.174 File Owner Class

1822 The property of a file indicating access permissions for a process related to the user
1823 identification of a process. A process is in the file owner class of a file if the effective user ID of
1824 the process matches the user ID of the file.

1825 3.175 File Permission Bits

1826 Information about a file that is used, along with other information, to determine whether a
1827 process has read, write, or execute/search permission to a file. The bits are divided into three
1828 parts: owner, group, and other. Each part is used with the corresponding file class of processes.
1829 These bits are contained in the file mode.

1830 **Note:** File modes are defined in detail in [<sys/stat.h>](#).

1831 File Access Permissions are defined in detail in [Section 4.5](#) (on page 108).

1832 3.176 File Serial Number

1833 A per-file system unique identifier for a file.

1834 3.177 File System

1835 A collection of files and certain of their attributes. It provides a name space for file serial
1836 numbers referring to those files.

1837 3.178 File Type

1838 See *File* in [Section 3.164](#) (on page 59).

1839 3.179 Filter

1840 A command whose operation consists of reading data from standard input or a list of input files
1841 and writing data to standard output. Typically, its function is to perform some transformation
1842 on the data stream.

1843 3.180 First Open (of a File)

1844 When a process opens a file that is not currently an open file within any process.

1845 3.181 Flow Control

1846 The mechanism employed by a communications provider that constrains a sending entity to
1847 wait until the receiving entities can safely receive additional data without loss.

1848 3.182 Foreground Job

1849 See *Foreground Process Group* in [Section 3.184](#).

1850 3.183 Foreground Process

1851 A process that is a member of a foreground process group.

1852 3.184 Foreground Process Group (or Foreground Job)

1853 A process group whose member processes have certain privileges, denied to processes in
1854 background process groups, when accessing their controlling terminal. Each session that has
1855 established a connection with a controlling terminal has at most one process group of the session
1856 as the foreground process group of that controlling terminal.

1857 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#).

1858 3.185 Foreground Process Group ID

1859 The process group ID of the foreground process group.

1860 3.186 Form-Feed Character (<form-feed>)

1861 A character that in the output stream indicates that printing should start on the next page of an
1862 output device. It is the character designated by '\f' in the C language. If the form-feed is not
1863 the first character of an output line, the result is unspecified. It is unspecified whether this
1864 character is the exact sequence transmitted to an output device by the system to accomplish the
1865 movement to the next page.

1866 3.187 Graphic Character

1867 A member of the **graph** character class of the current locale.

1868 **Note:** The **graph** character class is defined in detail in [Section 7.3.1](#) (on page 139).

1869 3.188 Group Database

1870 A system database that contains at least the following information for each group ID:

1871 Group name

1872 Numerical group ID

1873 List of users allowed in the group

1874 The list of users allowed in the group is used by the *newgrp* utility.

1875 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2017.

1876 3.189 Group ID

1877 A non-negative integer, which can be contained in an object of type **gid_t**, that is used to identify
1878 a group of system users. Each system user is a member of at least one group. When the identity
1879 of a group is associated with a process, a group ID value is referred to as a real group ID, an
1880 effective group ID, one of the supplementary group IDs, or a saved set-group-ID. The value
1881 (**gid_t**)-1 shall not be a valid group ID, but does have a defined use in some interfaces defined in
1882 this standard.

1883 3.190 Group Name

1884 A string that is used to identify a group; see also [Section 3.188](#). To be portable across conforming
1885 systems, the value is composed of characters from the portable filename character set. The
1886 <hyphen-minus> should not be used as the first character of a portable group name.

1887 3.191 Hard Limit

1888 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.
1889 A non-privileged process is restricted to only lowering its hard limit.

1890 3.192 Hard Link

1891 The relationship between two directory entries that represent the same file; see also [Section 3.130](#)
1892 (on page 53). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*
1893 function. This term is contrasted against symbolic link; see also [Section 3.381](#) (on page 95).

1894 3.193 Home Directory

1895 The directory specified by the *HOME* environment variable.

1896 3.194 Host Byte Order

1897 The arrangement of bytes in any integer type when using a specific machine architecture.

1898 **Note:** Two common methods of byte ordering are big-endian and little-endian. Big-endian is a format
1899 for storage of binary data in which the most significant byte is placed first, with the rest in
1900 descending order. Little-endian is a format for storage or transmission of binary data in which
1901 the least significant byte is placed first, with the rest in ascending order. See also [Section 4.10](#) (on
1902 page 110).

1903 3.195 Incomplete Line

1904 A sequence of one or more non-`<newline>` characters at the end of the file.

1905 3.196 Inf

1906 A value representing +infinity or a value representing -infinity that can be stored in a floating
1907 type. Not all systems support the Inf values.

1908 3.197 Instrumented Application

1909 An application that contains at least one call to the trace point function *posix_trace_event()*. Each
1910 process of an instrumented application has a mapping of trace event names to trace event type
1911 identifiers. This mapping is used by the trace stream that is created for that process.

1912 3.198 Interactive Shell

1913 A processing mode of the shell that is suitable for direct user interaction.

1914 3.199 Internationalization

1915 The provision within a computer program of the capability of making itself adaptable to the
1916 requirements of different native languages, local customs, and coded character sets.

1917 3.200 Interprocess Communication

1918 A functionality enhancement to add a high-performance, deterministic interprocess
1919 communication facility for local communication.

1920 3.201 Invoke

1921 To perform command search and execution actions, except that searching for shell functions and
1922 special built-in utilities is suppressed; see also [Section 3.155](#) (on page 58).

1923 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.1](#) (on page 2367).

1924 3.202 Job

1925 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all
1926 in the same process group.

1927 **Note:** See also XCU [Section 2.9.2](#) (on page 2368).

1928 3.203 Job Control

1929 A facility that allows users selectively to stop (suspend) the execution of processes and continue
1930 (resume) their execution at a later point. The user typically employs this facility via the
1931 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

1932 3.204 Job Control Job ID

1933 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in
1934 the following table:

1935 **Table 3-1** Job Control Job ID Formats

1936 Job Control 1937 Job ID	1938 Meaning
1938 %%	1939 Current job.
1939 %+	1940 Current job.
1940 %-	1941 Previous job.
1941 %n	1942 Job number <i>n</i> .
1942 %string	1943 Job whose command begins with <i>string</i> .
1943 %?string	Job whose command contains <i>string</i> .

1944 3.205 Last Close (of a File)

1945 When a process closes a file, resulting in the file not being an open file within any process.

1946 3.206 Line

1947 A sequence of zero or more non-`<newline>` characters plus a terminating `<newline>` character.

1948 3.207 Linger

1949 The period of time before terminating a connection, to allow outstanding data to be transferred.

1950 3.208 Link

1951 See *Directory Entry* in [Section 3.130](#) (on page 53).

1952 3.209 Link Count

1953 The number of directory entries that refer to a particular file.

1954 3.210 Live Process

1955 An address space with one or more threads executing within that address space, and the
1956 required system resources for those threads.

1957 **Note:** Many of the system resources defined by POSIX.1-2017 are shared among all of the threads
1958 within a process. These include the process ID, the parent process ID, process group ID, session
1959 membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-ID,
1960 supplementary group IDs, current working directory, root directory, file mode creation mask,
1961 and file descriptors.

1962 **3.211 Local Customs**

1963 The conventions of a geographical area or territory for such things as date, time, and currency
1964 formats.

1965 **3.212 Local Interprocess Communication (Local IPC)**

1966 The transfer of data between processes in the same system.

1967 **3.213 Locale**

1968 The definition of the subset of a user's environment that depends on language and cultural
1969 conventions.

1970 **Note:** Locales are defined in detail in [Chapter 7](#) (on page 135).

1971 **3.214 Localization**

1972 The process of establishing information within a computer system specific to the operation of
1973 particular native languages, local customs, and coded character sets.

1974 **3.215 Login**

1975 The unspecified activity by which a user gains access to the system. Each login is associated
1976 with exactly one login name.

1977 **3.216 Login Name**

1978 A user name that is associated with a login.

1979 **3.217 Map**

1980 To create an association between a page-aligned range of the address space of a process and
1981 some memory object, such that a reference to an address in that range of the address space

1982 results in a reference to the associated memory object. The mapped memory object is not
1983 necessarily memory-resident.

1984 **3.218 Marked Message**

1985 A STREAMS message on which a certain flag is set. Marking a message gives the application
1986 protocol-specific information. An application can use *ioctl()* to determine whether a given
1987 message is marked.

1988 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of POSIX.1-2017.

1989 **3.219 Matched**

1990 A state applying to a sequence of zero or more characters when the characters in the sequence
1991 correspond to a sequence of characters defined by a basic regular expression or extended regular
1992 expression pattern.

1993 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 181).

1994 **3.220 Memory Mapped Files**

1995 A facility to allow applications to access files as part of the address space.

1996 **3.221 Memory Object**

1997 One of:

1998 A file (see [Section 3.164](#), on page 59)

1999 A shared memory object (see [Section 3.346](#), on page 89)

2000 A typed memory object (see [Section 3.429](#), on page 102)

2001 When used in conjunction with *mmap()*, a memory object appears in the address space of the
2002 calling process.

2003 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of POSIX.1-2017.

2004 **3.222 Memory-Resident**

2005 The process of managing the implementation in such a way as to provide an upper bound on
2006 memory access times.

2007 3.223 Message

2008 In the context of programmatic message passing, information that can be transferred between
2009 processes or threads by being added to and removed from a message queue. A message consists
2010 of a fixed-size message buffer.

2011 3.224 Message Catalog

2012 In the context of providing natural language messages to the user, a file or storage area
2013 containing program messages, command prompts, and responses to prompts for a particular
2014 native language, territory, and codeset.

2015 3.225 Message Catalog Descriptor

2016 In the context of providing natural language messages to the user, a per-process unique value
2017 used to identify an open message catalog. A message catalog descriptor may be implemented
2018 using a file descriptor.

2019 3.226 Message Queue

2020 In the context of programmatic message passing, an object to which messages can be added and
2021 removed. Messages may be removed in the order in which they were added or in priority order.

2022 3.227 Mode

2023 A collection of attributes that specifies a file's type and its access permissions.

2024 **Note:** File Access Permissions are defined in detail in [Section 4.5](#) (on page 108).

2025 3.228 Monotonic Clock

2026 A clock measuring real time, whose value cannot be set via `clock_settime()` and which cannot
2027 have negative clock jumps.

2028 3.229 Mount Point

2029 Either the system root directory or a directory for which the *st_dev* field of structure **stat** differs
2030 from that of its parent directory.

2031 **Note:** The **stat** structure is defined in detail in [<sys/stat.h>](#).

2032 3.230 Multi-Character Collating Element

2033 A sequence of two or more characters that collate as an entity. For example, in some coded
2034 character sets, an accented character is represented by a non-spacing accent, followed by the
2035 letter. Other examples are the Spanish elements *ch* and *ll*.

2036 3.231 Multi-Threaded Library

2037 A library containing object files that were produced by compiling with *c99* using the flags
2038 output by *getconf* `POSIX_V7_THREADS_CFLAGS`, or by compiling using a non-standard utility
2039 with equivalent flags, and which makes use of interfaces that are only made available by *c99*
2040 when the `-l pthread` option is used or makes use of `SIGEV_THREAD` notifications.

2041 3.232 Multi-Threaded Process

2042 A process that contains more than one thread.

2043 3.233 Multi-Threaded Program

2044 A program whose executable file was produced by compiling with *c99* using the flags output by
2045 *getconf* `POSIX_V7_THREADS_CFLAGS`, and linking with *c99* using the flags output by *getconf*
2046 `POSIX_V7_THREADS_LDFLAGS` and the `-l pthread` option, or by compiling and linking using
2047 a non-standard utility with equivalent flags. Execution of a multi-threaded program initially
2048 creates a single-threaded process; the process can create additional threads using *pthread_create()*
2049 or `SIGEV_THREAD` notifications.

2050 3.234 Mutex

2051 A synchronization object used to allow multiple threads to serialize their access to shared data.
2052 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has
2053 locked a mutex becomes its owner and remains the owner until that same thread unlocks the
2054 mutex.

2055 3.235 Name

2056 In the shell command language, a word consisting solely of underscores, digits, and alphabets
2057 from the portable character set. The first character of a name is not a digit.

2058 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

2059 3.236 Named STREAM

2060 A STREAMS-based file descriptor that is attached to a name in the file system name space. All
2061 subsequent operations on the named STREAM act on the STREAM that was associated with the
2062 file descriptor until the name is disassociated from the STREAM.

2063 3.237 NaN (Not a Number)

2064 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-
2065 point numbers. Not all systems support NaN values.

2066 3.238 Native Language

2067 A computer user's spoken or written language, such as American English, British English,
2068 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.

2069 3.239 Negative Response

2070 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
2071 keyword **noexpr**, matching an extended regular expression in the current locale.

2072 **Note:** The *LC_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 165).

2073 3.240 Network

2074 A collection of interconnected hosts.

2075 **Note:** The term "network" in POSIX.1-2017 is used to refer to the network of hosts. The term "batch
2076 system" is used to refer to the network of batch servers.

2077 3.241 Network Address

2078 A network-visible identifier used to designate specific endpoints in a network. Specific
2079 endpoints on host systems have addresses, and host systems may also have addresses.

2080 3.242 Network Byte Order

2081 The way of representing any integer type such that, when transmitted over a network via a
2082 network endpoint, the **int** type is transmitted as an appropriate number of octets with the most
2083 significant octet first, followed by any other octets in descending order of significance.

2084 **Note:** This order is more commonly known as big-endian ordering. See also [Section 4.10](#) (on page
2085 110).

2086 3.243 Newline Character (<newline>)

2087 A character that in the output stream indicates that printing should start at the beginning of the
2088 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this
2089 character is the exact sequence transmitted to an output device by the system to accomplish the
2090 movement to the next line.

2091 3.244 Nice Value

2092 A number used as advice to the system to alter process scheduling. Numerically smaller values
2093 give a process additional preference when scheduling a process to run. Numerically larger
2094 values reduce the preference and make a process less likely to run. Typically, a process with a
2095 smaller nice value runs to completion more quickly than an equivalent process with a higher
2096 nice value. The symbol {NZERO} specifies the default nice value of the system.

2097 3.245 Non-Blocking

2098 A property of an open file description that causes function calls involving it to return without
2099 delay when it is detected that the requested action associated with the function call cannot be
2100 completed without unknown delay.

2101 **Note:** The exact semantics are dependent on the type of file associated with the open file description.
2102 For data reads from devices such as ttys and FIFOs, this property causes the read to return
2103 immediately when no data was available. Similarly, for writes, it causes the call to return
2104 immediately when the thread would otherwise be delayed in the write operation; for example,
2105 because no space was available. For networking, it causes functions not to await protocol events
2106 (for example, acknowledgements) to occur. See also XSH [Section 2.10.7](#) (on page 525).

2107 3.246 Non-Spacing Characters

2108 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:2001
2109 standard coded graphic character set, which is used in combination with other characters to
2110 form composite graphic symbols.

2111 3.247 NUL

2112 A character with all bits set to zero.

2113 3.248 Null Byte

2114 A byte with all bits set to zero.

2115 3.249 Null Pointer

2116 A pointer obtained by converting an integer constant expression with the value 0, or such an
2117 expression cast to type **void ***, to a pointer type; for example, **(char *)0**. The C language
2118 guarantees that a null pointer compares unequal to a pointer to any object or function, so it is
2119 used by many functions that return pointers to indicate an error.

2120 3.250 Null String

2121 See *Empty String* in [Section 3.146](#) (on page 56).

2122 3.251 Null Wide-Character Code

2123 A wide-character code with all bits set to zero.

2124 3.252 Number-Sign Character (<number-sign>)

2125 The character '#', also known as hash sign.

2126 3.253 Object File

2127 A regular file containing the output of a compiler, formatted as input to a linkage editor for
2128 linking with other object files into an executable form. The methods of linking are unspecified
2129 and may involve the dynamic linking of objects at runtime. The internal format of an object file
2130 is unspecified, but a conforming application cannot assume an object file is a text file.

2131 3.254 Octet

2132 Unit of data representation that consists of eight contiguous bits.

2133 3.255 Offset Maximum

2134 An attribute of an open file description representing the largest value that can be used as a file
2135 offset.

2136 3.256 Opaque Address

2137 An address such that the entity making use of it requires no details about its contents or format.

2138 3.257 Open File

2139 A file that is currently associated with a file descriptor.

2140 3.258 Open File Description

2141 A record of how a process or group of processes is accessing a file. Each file descriptor refers to
2142 exactly one open file description, but an open file description can be referred to by more than
2143 one file descriptor. The file offset, file status, and file access modes are attributes of an open file
2144 description.

2145 3.259 Operand

2146 An argument to a command that is generally used as an object supplying information to a utility
2147 necessary to complete its processing. Operands generally follow the options in a command line.

2148 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2149 3.260 Operator

2150 In the shell command language, either a control operator or a redirection operator.

2151 **3.261 Option**

2152 An argument to a command that is generally used to specify changes in the utility's default
2153 behavior.

2154 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2155 **3.262 Option-Argument**

2156 A parameter that follows certain options. In some cases an option-argument is included within
2157 the same argument string as the option—in most cases it is the next argument.

2158 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 213).

2159 **3.263 Orientation**

2160 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2161 **Note:** For further information, see XSH [Section 2.5.2](#) (on page 498).

2162 **3.264 Orphaned Process Group**

2163 A process group in which the parent of every member is either itself a member of the group or is
2164 not a member of the group's session.

2165 **3.265 Page**

2166 The granularity of process memory mapping or locking.

2167 Physical memory and memory objects can be mapped into the address space of a process on
2168 page boundaries and in integral multiples of pages. Process address space can be locked into
2169 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2170 **3.266 Page Size**

2171 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On
2172 systems that have segment rather than page-based memory architectures, the term "page"
2173 means a segment.

2174 3.267 Parameter

2175 In the shell command language, an entity that stores values. There are three types of parameters:
 2176 variables (named parameters), positional parameters, and special parameters. Parameter
 2177 expansion is accomplished by introducing a parameter with the '\$' character.

2178 **Note:** See also XCU [Section 2.5](#) (on page 2349).

2179 In the C language, an object declared as part of a function declaration or definition that acquires
 2180 a value on entry to the function, or an identifier following the macro name in a function-like
 2181 macro definition.

2182 3.268 Parent Directory

2183 When discussing a given directory, the directory that both contains a directory entry for the
 2184 given directory and is represented by the pathname dot-dot in the given directory.

2185 When discussing other types of files, a directory containing a directory entry for the file under
 2186 discussion.

2187 This concept does not apply to dot and dot-dot.

2188 3.269 Parent Process

2189 The process which created (or inherited) the process under discussion.

2190 3.270 Parent Process ID

2191 An attribute of a new process identifying the parent of the process. The parent process ID of a
 2192 process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime
 2193 has ended, the parent process ID is the process ID of an implementation-defined system process.

2194 3.271 Pathname

2195 A string that is used to identify a file. In the context of POSIX.1-2017, a pathname may be limited
 2196 to {PATH_MAX} bytes, including the terminating null byte. It has optional beginning <slash>
 2197 characters, followed by zero or more filenames separated by <slash> characters. A pathname
 2198 can optionally contain one or more trailing <slash> characters. Multiple successive <slash>
 2199 characters are considered to be the same as one <slash>, except for the case of exactly two
 2200 leading <slash> characters.

2201 **Note:** If a pathname consists of only bytes corresponding to characters from the portable filename
 2202 character set (see [Section 3.282](#), on page 79), <slash> characters, and a single terminating
 2203 <NUL> character, the pathname will be usable as a character string in all supported locales;
 2204 otherwise, the pathname might only be a string (rather than a character string). Additionally,
 2205 since the single-byte encoding of the <slash> character is required to be the same across all
 2206 locales and to not occur within a multi-byte character, references to a <slash> character within a
 2207 pathname are well-defined even when the pathname is not a character string. However, this
 2208 property does not necessarily hold for the remaining characters within the portable filename

2209 character set.

2210 Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

2211 **3.272 Pathname Component**

2212 See *Filename* in [Section 3.170](#) (on page 60).

2213 **3.273 Path Prefix**

2214 The part of a pathname up to, but not including, the last component and any trailing <slash>
2215 characters, unless the pathname consists entirely of <slash> characters, in which case the path
2216 prefix is '/' for a pathname containing either a single <slash> or three or more <slash>
2217 characters, and '//' for the pathname //. The path prefix of a pathname containing no <slash>
2218 characters is empty, but is treated as referring to the current working directory.

2219 **Note:** The term is used both in the sense of identifying part of a pathname that forms the prefix and of
2220 joining a non-empty path prefix to a filename to form a pathname. In the latter case, the path
2221 prefix need not have a trailing <slash> (in which case the joining is done with a <slash>
2222 character).

2223 **3.274 Pattern**

2224 A sequence of characters used either with regular expression notation or for pathname
2225 expansion, as a means of selecting various character strings or pathnames, respectively.

2226 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 181).

2227 See also XCU [Section 2.6.6](#) (on page 2360).

2228 The syntaxes of the two types of patterns are similar, but not identical; POSIX.1-2017 always
2229 indicates the type of pattern being referred to in the immediate context of the use of the term.

2230 **3.275 Period Character (<period>)**

2231 The character '.'. The term ``period'' is contrasted with dot (see also [Section 3.136](#), on page 54),
2232 which is used to describe a specific directory entry.

2233 3.276 Permissions

2234 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2235 **Note:** File Access Permissions are defined in detail in [Section 4.5](#) (on page 108).

2236 3.277 Persistence

2237 A mode for semaphores, shared memory, and message queues requiring that the object and its
2238 state (including data, if any) are preserved after the object is no longer referenced by any
2239 process.

2240 Persistence of an object does not imply that the state of the object is maintained across a system
2241 crash or a system reboot.

2242 3.278 Pipe

2243 An object identical to a FIFO which has no links in the file hierarchy.

2244 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of POSIX.1-2017.

2245 3.279 Polling

2246 A scheduling scheme whereby the local process periodically checks until the pre-specified
2247 events (for example, read, write) have occurred.

2248 3.280 Portable Character Set

2249 The collection of characters that are required to be present in all locales supported by
2250 conforming systems.

2251 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 125).

2252 This term is contrasted against the smaller portable filename character set; see also [Section 3.282](#)
2253 (on page 79).

2254 3.281 Portable Filename

2255 A filename consisting only of characters from the portable filename character set.

2256 **Note:** Applications should avoid using filenames that have the <hyphen-minus> character as the first
2257 character since this may cause problems when filenames are passed as command line
2258 arguments.

2259 3.282 Portable Filename Character Set

2260 The set of characters from which portable filenames are constructed.

2261 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
2262 a b c d e f g h i j k l m n o p q r s t u v w x y z
2263 0 1 2 3 4 5 6 7 8 9 . _ -

2264 The last three characters are the <period>, <underscore>, and <hyphen-minus> characters,
2265 respectively. See also [Section 3.271](#) (on page 76).

2266 3.283 Positional Parameter

2267 In the shell command language, a parameter denoted by a single digit or one or more digits in
2268 curly braces.

2269 **Note:** For further information, see XCU [Section 2.5.1](#) (on page 2349).

2270 3.284 Preallocation

2271 The reservation of resources in a system for a particular use.

2272 Preallocation does not imply that the resources are immediately allocated to that use, but merely
2273 indicates that they are guaranteed to be available in bounded time when needed.

2274 3.285 Preempted Process (or Thread)

2275 A running thread whose execution is suspended due to another thread becoming runnable at a
2276 higher priority.

2277 3.286 Previous Job

2278 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
2279 current job exits. There is at most one previous job; see also [Section 3.204](#) (on page 66).

2280 3.287 Printable Character

2281 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
2282 the current locale.

2283 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

2284 3.288 Printable File

2285 A text file consisting only of the characters included in the **print** and **space** character
2286 classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

2287 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 139).

2288 3.289 Priority

2289 A non-negative integer associated with processes or threads whose value is constrained to a
2290 range defined by the applicable scheduling policy. Numerically higher values represent higher
2291 priorities.

2292 3.290 Priority Band

2293 The queuing order applied to normal priority STREAMS messages. High priority STREAMS
2294 messages are not grouped by priority bands. The only differentiation made by the STREAMS
2295 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate
2296 between priority bands.

2297 3.291 Priority Inversion

2298 A condition in which a thread that is not voluntarily suspended (waiting for an event or time
2299 delay) is not running while a lower priority thread is running. Such blocking of the higher
2300 priority thread is often caused by contention for a shared resource.

2301 3.292 Priority Scheduling

2302 A performance and determinism improvement facility to allow applications to determine the
2303 order in which threads that are ready to run are granted access to processor resources.

2304 3.293 Priority-Based Scheduling

2305 Scheduling in which the selection of a running thread is determined by the priorities of the
2306 runnable processes or threads.

2307 **3.294 Privilege**

2308 See *Appropriate Privileges* in [Section 3.20](#) (on page 36).

2309 **3.295 Process**

2310 A live process (see [Section 3.210](#), on page 66) or a zombie process (see [Section 3.452](#), on page
2311 106). The lifetime of a process is described in [Section 3.301](#) (on page 82).

2312 **3.296 Process Group**

2313 A collection of processes that permits the signaling of related processes. Each process in the
2314 system is a member of a process group that is identified by a process group ID. A newly created
2315 process joins the process group of its creator.

2316 **3.297 Process Group ID**

2317 The unique positive integer identifier representing a process group during its lifetime.

2318 **Note:** See also *Process Group ID Reuse* defined in [Section 4.14](#) (on page 113).

2319 **3.298 Process Group Leader**

2320 A process whose process ID is the same as its process group ID.

2321 **3.299 Process Group Lifetime**

2322 The period of time that begins when a process group is created and ends when the last
2323 remaining process in the group leaves the group, due either to the end of the lifetime of the last
2324 process or to the last remaining process calling the *setsid()* or *setpgid()* functions.

2325 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of
2326 POSIX.1-2017.

2327 3.300 Process ID

2328 The unique positive integer identifier representing a process during its lifetime.

2329 **Note:** See also *Process ID Reuse* defined in [Section 4.14](#) (on page 113).

2330 3.301 Process Lifetime

2331 The period of time that begins when a process is created and ends when its process ID is
2332 returned to the system.

2333 See also [Section 3.210](#) (on page 66), [Section 3.303](#), and [Section 3.452](#) (on page 106).

2334 **Note:** Process creation is defined in detail in the descriptions of the *fork()*, *posix_spawn()*, and
2335 *posix_spawnp()* functions in the System Interfaces volume of POSIX.1-2017.

2336 3.302 Process Memory Locking

2337 A performance improvement facility to bind application programs into the high-performance
2338 random access memory of a computer system. This avoids potential latencies introduced by the
2339 operating system in storing parts of a program that were not recently referenced on secondary
2340 memory devices.

2341 3.303 Process Termination

2342 There are two kinds of process termination:

- 2343 1. Normal termination occurs by a return from *main()*, when requested with the *exit()*,
2344 *_exit()*, or *_Exit()* functions; or when the last thread in the process terminates by
2345 returning from its start function, by calling the *pthread_exit()* function, or through
2346 cancellation.
- 2347 2. Abnormal termination occurs when requested by the *abort()* function or when some
2348 signals are received.

2349 **Note:** The consequences of process termination can be found in the description of the *_Exit()* function
2350 in the System Interfaces volume of POSIX.1-2017. The *_exit()*, *_Exit()*, *abort()*, and *exit()*
2351 functions are defined in detail in the System Interfaces volume of POSIX.1-2017.

2352 3.304 Process-To-Process Communication

2353 The transfer of data between processes.

2354 3.305 Process Virtual Time

2355 The measurement of time in units elapsed by the system clock while a process is executing.

2356 3.306 Program

2357 A prepared sequence of instructions to the system to accomplish a defined task. The term
2358 “program” in POSIX.1-2017 encompasses applications written in the Shell Command Language,
2359 complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level languages.

2360 3.307 Protocol

2361 A set of semantic and syntactic rules for exchanging information.

2362 3.308 Pseudo-Terminal

2363 A facility that provides an interface that is identical to the terminal subsystem, except where
2364 noted otherwise in POSIX.1-2017. A pseudo-terminal is composed of two devices: the “master
2365 device” and a “slave device”. The slave device provides processes with an interface that is
2366 identical to the terminal interface, although there need not be hardware behind that interface.
2367 Anything written on the master device is presented to the slave as an input and anything
2368 written on the slave device is presented as an input on the master side.

2369 3.309 Radix Character

2370 The character that separates the integer part of a number from the fractional part.

2371 3.310 Read-Only File System

2372 A file system that has implementation-defined characteristics restricting modifications.

2373 **Note:** File Times Update is described in detail in [Section 4.9](#) (on page 109).

2374 3.311 Read-Write Lock

2375 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2376 read-only access to data while allowing only one thread to have write access at any given time.
2377 They are typically used to protect data that is read-only more frequently than it is changed.

2378 Read-write locks can be used to synchronize threads in the current process and other processes if
2379 they are allocated in memory that is writable and shared among the cooperating processes and
2380 have been initialized for this behavior.

2381 **3.312 Real Group ID**

2382 The attribute of a process that, at the time of process creation, identifies the group of the user
2383 who created the process; see also [Section 3.189](#) (on page 63).

2384 **3.313 Real Time**

2385 Time measured as total units elapsed by the system clock without regard to which thread is
2386 executing.

2387 **3.314 Realtime Signal Extension**

2388 A determinism improvement facility to enable asynchronous signal notifications to an
2389 application to be queued without impacting compatibility with the existing signal functions.

2390 **3.315 Real User ID**

2391 The attribute of a process that, at the time of process creation, identifies the user who created the
2392 process; see also [Section 3.436](#) (on page 103).

2393 **3.316 Record**

2394 A collection of related data units or words which is treated as a unit.

2395 **3.317 Redirection**

2396 In the shell command language, a method of associating files with the input or output of
2397 commands.

2398 **Note:** For further information, see XCU [Section 2.7](#) (on page 2360).

2399 3.318 Redirection Operator

2400 In the shell command language, a token that performs a redirection function. It is one of the
2401 following symbols:

2402 < > >| << >> <& >& <<- <>

2403 3.319 Referenced Shared Memory Object

2404 A shared memory object that is open or has one or more mappings defined on it.

2405 3.320 Refresh

2406 To ensure that the information on the user's terminal screen is up-to-date.

2407 3.321 Regular Expression

2408 A pattern that selects specific strings from a set of character strings.

2409 **Note:** Regular Expressions are described in detail in [Chapter 9](#) (on page 181).

2410 3.322 Region

2411 In the context of the address space of a process, a sequence of addresses.

2412 In the context of a file, a sequence of offsets.

2413 3.323 Regular File

2414 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
2415 system.

2416 3.324 Relative Pathname

2417 A pathname not beginning with a <slash> character.

2418 **Note:** Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

2419 3.325 Relocatable File

2420 A file holding code or data suitable for linking with other object files to create an executable or a
2421 shared object file.

2422 3.326 Relocation

2423 The process of connecting symbolic references with symbolic definitions. For example, when a
2424 program calls a function, the associated call instruction transfers control to the proper
2425 destination address at execution.

2426 3.327 Requested Batch Service

2427 A service that is either rejected or performed prior to a response from the service to the
2428 requester.

2429 3.328 (Time) Resolution

2430 The minimum time interval that a clock can measure or whose passage a timer can detect.

2431 3.329 Robust Mutex

2432 A mutex with the *robust* attribute set.

2433 **Note:** The *robust* attribute is defined in detail by the `pthread_mutexattr_getrobust()` function.

2434 3.330 Root Directory

2435 A directory, associated with a process, that is used in pathname resolution for pathnames that
2436 begin with a <slash> character.

2437 3.331 Runnable Process (or Thread)

2438 A thread that is capable of being a running thread, but for which no processor is available.

2439 3.332 Running Process (or Thread)

2440 A thread currently executing on a processor. On multi-processor systems there may be more
2441 than one such thread in a system at a time.

2442 3.333 Saved Resource Limits

2443 An attribute of a process that provides some flexibility in the handling of unrepresentable
2444 resource limits, as described in the *exec* family of functions and *setrlimit()*.

2445 **Note:** The *exec* and *setrlimit()* functions are defined in detail in the System Interfaces volume of
2446 POSIX.1-2017.

2447 3.334 Saved Set-Group-ID

2448 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2449 attribute, as described in the *exec* family of functions and *setgid()*.

2450 **Note:** The *exec* and *setgid()* functions are defined in detail in the System Interfaces volume of
2451 POSIX.1-2017.

2452 3.335 Saved Set-User-ID

2453 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2454 attribute, as described in the *exec* family of functions and *setuid()*.

2455 **Note:** The *exec* and *setuid()* functions are defined in detail in the System Interfaces volume of
2456 POSIX.1-2017.

2457 3.336 Scheduling

2458 The application of a policy to select a runnable process or thread to become a running process or
2459 thread, or to alter one or more of the thread lists.

2460 3.337 Scheduling Allocation Domain

2461 The set of processors on which an individual thread can be scheduled at any given time.

2462 3.338 Scheduling Contention Scope

2463 A property of a thread that defines the set of threads against which that thread competes for
2464 resources.

2465 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
2466 processor resources. In POSIX.1-2017, a thread has scheduling contention scope of either
2467 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

2468 **3.339 Scheduling Policy**

2469 A set of rules that is used to determine the order of execution of processes or threads to achieve
2470 some goal.

2471 **Note:** Scheduling Policy is defined in detail in [Section 4.15](#) (on page 113).

2472 **3.340 Screen**

2473 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a
2474 physical display device or may occupy the entire physical area of the display device.

2475 **3.341 Scroll**

2476 To move the representation of data vertically or horizontally relative to the terminal screen.
2477 There are two types of scrolling:

- 2478 1. The cursor moves with the data.
- 2479 2. The cursor remains stationary while the data moves.

2480 **3.342 Semaphore**

2481 A minimum synchronization primitive to serve as a basis for more complex synchronization
2482 mechanisms to be defined by the application program.

2483 **Note:** Semaphores are defined in detail in [Section 4.17](#) (on page 114).

2484 3.343 Session

2485 A collection of process groups established for job control purposes. Each process group is a
2486 member of a session. A process is considered to be a member of the session of which its process
2487 group is a member. A newly created process joins the session of its creator. A process can alter
2488 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2489 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of POSIX.1-2017.

2490 3.344 Session Leader

2491 A process that has created a session.

2492 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of
2493 POSIX.1-2017.

2494 3.345 Session Lifetime

2495 The period between when a session is created and the end of the lifetime of all the process
2496 groups that remain as members of the session.

2497 3.346 Shared Memory Object

2498 An object that represents memory that can be mapped concurrently into the address space of
2499 more than one process.

2500 3.347 Shell

2501 A program that interprets sequences of text input as commands. It may operate on an input
2502 stream or it may interactively prompt and read commands from a terminal.

2503 3.348 Shell, the

2504 The Shell Command Language Interpreter; a specific instance of a shell.

2505 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
2506 POSIX.1-2017.

2507 3.349 Shell Script

2508 A file containing shell commands. If the file is made executable, it can be executed by specifying
2509 its name as a simple command. Execution of a shell script causes a shell to execute the
2510 commands within the script. Alternatively, a shell can be requested to execute the commands in
2511 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2512 **Note:** Simple Commands are defined in detail in XCU [Section 2.9.1](#) (on page 2365).

2513 The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2017.

2514 3.350 Signal

2515 A mechanism by which a process or thread may be notified of, or affected by, an event occurring
2516 in the system. Examples of such events include hardware exceptions and specific actions by
2517 processes. The term signal is also used to refer to the event itself.

2518 3.351 Signal Stack

2519 Memory established for a thread, in which signal handlers catching signals sent to that thread
2520 are executed.

2521 3.352 Single-Quote Character

2522 The character designated by ' \ ' ' in the C language, also known as <apostrophe>.

2523 3.353 Single-Threaded Process

2524 A process that contains a single thread.

2525 3.354 Single-Threaded Program

2526 A program whose executable file was produced by compiling with *c99* without using the flags
2527 output by *getconf* POSIX_V7_THREADS_CFLAGS and linking with *c99* using neither the flags
2528 output by *getconf* POSIX_V7_THREADS_LDFLAGS nor the **-l pthread** option, or by compiling
2529 and linking using a non-standard utility with equivalent flags. Execution of a single-threaded
2530 program creates a single-threaded process; if the process attempts to create additional threads
2531 using *pthread_create()* or SIGEV_THREAD notifications, the behavior is undefined. If the process
2532 uses *dlopen()* to load a multi-threaded library, the behavior is undefined.

2533 3.355 Slash Character (<slash>)

2534 The character ' / ', also known as solidus.

2535 3.356 Socket

2536 A file of a particular type that is used as a communications endpoint for process-to-process
2537 communication as described in the System Interfaces volume of POSIX.1-2017.

2538 3.357 Socket Address

2539 An address associated with a socket or remote endpoint, including an address family identifier
2540 and addressing information specific to that address family. The address may include multiple
2541 parts, such as a network address associated with a host system and an identifier for a specific
2542 endpoint.

2543 3.358 Soft Limit

2544 A resource limitation established for each process that the process may set to any value less than
2545 or equal to the hard limit.

2546 3.359 Source Code

2547 When dealing with the Shell Command Language, input to the command language interpreter.
2548 The term “shell script” is synonymous with this meaning.

2549 When dealing with an ISO/IEC-conforming programming language, source code is input to a
2550 compiler conforming to that ISO/IEC standard.

2551 Source code also refers to the input statements prepared for the following standard utilities: *awk*,
2552 *bc*, *ed*, *ex*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

2553 Source code can also refer to a collection of sources meeting any or all of these meanings.

2554 **Note:** The *awk*, *bc*, *ed*, *ex*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and
2555 Utilities volume of POSIX.1-2017.

2556 3.360 Space Character (<space>)

2557 The character defined in the portable character set as <space>. The <space> character is a
2558 member of the **space** character class of the current locale, but represents the single character, and
2559 not all of the possible members of the class; see also [Section 3.442](#) (on page 104).

2560 3.361 Spawn

2561 A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient
2562 replacement for *fork()/exec*.

2563 3.362 Special Built-In

2564 See *Built-In Utility* in [Section 3.83](#) (on page 46).

2565 3.363 Special Parameter

2566 In the shell command language, a parameter named by a single character from the following list:

2567 * @ # ? ! - \$ 0

2568 **Note:** For further information, see XCU [Section 2.5.2](#) (on page 2350).

2569 3.364 Spin Lock

2570 A synchronization object used to allow multiple threads to serialize their access to shared data.

2571 3.365 Sporadic Server

2572 A scheduling policy for threads and processes that reserves a certain amount of execution
2573 capacity for processing aperiodic events at a given priority level.

2574 3.366 Standard Error

2575 An output stream usually intended to be used for diagnostic messages.

2576 3.367 Standard Input

2577 An input stream usually intended to be used for primary data input.

2578 3.368 Standard Output

2579 An output stream usually intended to be used for primary data output.

2580 3.369 Standard Utilities

2581 The utilities described in the Shell and Utilities volume of POSIX.1-2017.

2582 3.370 Stream

2583 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
2584 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
2585 *fmemopen()*, *fopen()*, *open_memstream()*, or *popen()* functions, and are associated with a file
2586 descriptor. A stream provides the additional services of user-selectable buffering and formatted
2587 input and output; see also [Section 3.371](#).

2588 **Note:** For further information, see XSH [Section 2.5](#) (on page 495).

2589 The *fdopen()*, *fmemopen()*, *fopen()*, *open_memstream()*, and *popen()* functions are defined in detail
2590 in the System Interfaces volume of POSIX.1-2017.

2591 3.371 STREAM

2592 Appearing in uppercase, STREAM refers to a full-duplex connection between a process and an
2593 open device or pseudo-device. It optionally includes one or more intermediate processing
2594 modules that are interposed between the process end of the STREAM and the device driver (or
2595 pseudo-device driver) end of the STREAM; see also [Section 3.370](#).

2596 **Note:** For further information, see XSH [Section 2.6](#) (on page 500).

2597 3.372 STREAM End

2598 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
2599 the STREAM.

2600 3.373 STREAM Head

2601 The STREAM head is the beginning of the STREAM and is at the boundary between the system
2602 and the application process. This is also known as the upstream end of the STREAM.

2603 3.374 STREAMS Multiplexor

2604 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected
2605 above is referred to as N-to-1, or “upper multiplexing”. Multiplexing with STREAMS connected
2606 below is referred to as 1-to-N or “lower multiplexing”.

2607 3.375 String

2608 A contiguous sequence of bytes terminated by and including the first null byte.

2609 3.376 Subshell

2610 A shell execution environment, distinguished from the main or current shell execution
2611 environment.

2612 **Note:** For further information, see XCU [Section 2.12](#) (on page 2381).

2613 3.377 Successfully Transferred

2614 For a write operation to a regular file, when the system ensures that all data written is readable
2615 on any subsequent open of the file (even one that follows a system or power failure) in the
2616 absence of a failure of the physical storage medium.

2617 For a read operation, when an image of the data on the physical storage medium is available to
2618 the requesting process.

2619 3.378 Supplementary Group ID

2620 An attribute of a process used in determining file access permissions. A process has up to
2621 {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The
2622 supplementary group IDs of a process are set to the supplementary group IDs of the parent
2623 process when the process is created.

2624 3.379 Suspended Job

2625 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the
2626 process group to stop. A suspended job is a background job, but a background job is not
2627 necessarily a suspended job.

2628 **3.380 Symbolic Constant**

2629 An object-like macro defined with a constant value.

2630 Unless stated otherwise, the following shall apply to every symbolic constant:

2631 It expands to a compile-time constant expression with an integer type.

2632 It may be defined as another type of constant *†*e.g., an enumeration constant *†*as well as
2633 being a macro.

2634 It need not be usable in `#if` preprocessing directives.

2635 **3.381 Symbolic Link**

2636 A type of file with the property that when the file is encountered during pathname resolution, a
2637 string stored by the file is used to modify the pathname resolution. The stored string has a
2638 length of {SYMLINK_MAX} bytes or fewer.

2639 **Note:** Pathname Resolution is defined in detail in [Section 4.13](#) (on page 111).

2640 **3.382 Synchronized Input and Output**

2641 A determinism and robustness improvement mechanism to enhance the data input and output
2642 mechanisms, so that an application can ensure that the data being manipulated is physically
2643 present on secondary mass storage devices.

2644 **3.383 Synchronized I/O Completion**

2645 The state of an I/O operation that has either been successfully transferred or diagnosed as
2646 unsuccessful.

2647 **3.384 Synchronized I/O Data Integrity Completion**

2648 For read, when the operation has been completed or diagnosed if unsuccessful. The read is
2649 complete only when an image of the data has been successfully transferred to the requesting
2650 process. If there were any pending write requests affecting the data to be read at the time that
2651 the synchronized read operation was requested, these write requests are successfully transferred
2652 prior to reading the data.

2653 For write, when the operation has been completed or diagnosed if unsuccessful. The write is
2654 complete only when the data specified in the write request is successfully transferred and all file
2655 system information required to retrieve the data is successfully transferred.

2656 File attributes that are not necessary for data retrieval (access time, modification time, status
2657 change time) need not be successfully transferred prior to returning to the calling process.

2658 3.385 Synchronized I/O File Integrity Completion

2659 Identical to a synchronized I/O data integrity completion with the addition that all file
2660 attributes relative to the I/O operation (including access time, modification time, status change
2661 time) are successfully transferred prior to returning to the calling process.

2662 3.386 Synchronized I/O Operation

2663 An I/O operation performed on a file that provides the application assurance of the integrity of
2664 its data and files.

2665 3.387 Synchronous I/O Operation

2666 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the
2667 processor until that I/O operation completes.

2668 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or
2669 synchronized I/O file integrity completion.

2670 3.388 Synchronously-Generated Signal

2671 A signal that is attributable to a specific thread.

2672 For example, a thread executing an illegal instruction or touching invalid memory causes a
2673 synchronously-generated signal. Being synchronous is a property of how the signal was
2674 generated and not a property of the signal number.

2675 3.389 System

2676 An implementation of POSIX.1-2017.

2677 3.390 System Boot

2678 An unspecified sequence of events that may result in the loss of transitory data; that is, data that
2679 is not saved in permanent storage. For example, message queues, shared memory, semaphores,
2680 and processes.

2681 3.391 System Clock

2682 A clock with at least one second resolution that contains seconds since the Epoch.

2683 3.392 System Console

2684 A device that receives messages sent by the *syslog()* function, and the *fmtmsg()* function when
2685 the MM_CONSOLE flag is set.

2686 **Note:** The *syslog()* and *fmtmsg()* functions are defined in detail in the System Interfaces volume of
2687 POSIX.1-2017.

2688 3.393 System Crash

2689 An interval initiated by an unspecified circumstance that causes all processes (possibly other
2690 than special system processes) to be terminated in an undefined manner, after which any
2691 changes to the state and contents of files created or written to by an application prior to the
2692 interval are undefined, except as required elsewhere in POSIX.1-2017.

2693 3.394 System Databases

2694 An implementation provides two system databases: the “group database” (see also [Section](#)
2695 [3.188](#), on page 63) and the “user database” (see also [Section 3.435](#), on page 103).

2696 3.395 System Documentation

2697 All documentation provided with an implementation except for the conformance document.
2698 Electronically distributed documents for an implementation are considered part of the system
2699 documentation.

2700 3.396 System Process

2701 An object other than a process executing an application, that is provided by the system and has a
2702 process ID.

2703 3.397 System Reboot

2704 See *System Boot* defined in [Section 3.390](#) (on page 96).

2705 3.398 System Trace Event

2706 A trace event that is generated by the implementation, in response either to a system-initiated
2707 action or to an application-requested action, except for a call to *posix_trace_event()*. When
2708 supported by the implementation, a system-initiated action generates a process-independent
2709 system trace event and an application-requested action generates a process-dependent system
2710 trace event. For a system trace event not defined by POSIX.1-2017, the associated trace event

2711 type identifier is derived from the implementation-defined name for this trace event, and the
2712 associated data is of implementation-defined content and length.

2713 **3.399 System-Wide**

2714 Pertaining to events occurring in all processes existing in an implementation at a given point in
2715 time.

2716 **3.400 Tab Character (<tab>)**

2717 A character that in the output stream indicates that printing or displaying should start at the
2718 next horizontal tabulation position on the current line. It is the character designated by '\t' in
2719 the C language. If the current position is at or past the last defined horizontal tabulation
2720 position, the behavior is unspecified. It is unspecified whether this character is the exact
2721 sequence transmitted to an output device by the system to accomplish the tabulation.

2722 **3.401 Terminal (or Terminal Device)**

2723 A character special file that obeys the specifications of the general terminal interface.

2724 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 199).

2725 **3.402 Text Column**

2726 A roughly rectangular block of characters capable of being laid out side-by-side next to other
2727 text columns on an output page or terminal screen. The widths of text columns are measured in
2728 column positions.

2729 **3.403 Text File**

2730 A file that contains characters organized into zero or more lines. The lines do not contain NUL
2731 characters and none can exceed {LINE_MAX} bytes in length, including the <newline>
2732 character. Although POSIX.1-2017 does not distinguish between text files and binary files (see
2733 the ISO C standard), many utilities only produce predictable or meaningful output when
2734 operating on text files. The standard utilities that have such restrictions always specify ``text
2735 files'' in their STDIN or INPUT FILES sections.

2736 3.404 Thread

2737 A single flow of control within a process. Each thread has its own thread ID, scheduling priority
 2738 and policy, *errno* value, floating point environment, thread-specific key/value bindings, and the
 2739 required system resources to support a flow of control. Anything whose address may be
 2740 determined by a thread, including but not limited to static variables, storage obtained via
 2741 *malloc()*, directly addressable storage obtained through implementation-defined functions, and
 2742 automatic variables, are accessible to all threads in the same process.

2743 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of POSIX.1-2017.

2744 3.405 Thread ID

2745 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t**
 2746 called a thread ID.

2747 3.406 Thread List

2748 An ordered set of runnable threads that all have the same ordinal value for their priority.

2749 The ordering of threads on the list is determined by a scheduling policy or policies. The set of
 2750 thread lists includes all runnable threads in the system.

2751 3.407 Thread-Safe

2752 A thread-safe function can be safely invoked concurrently with other calls to the same function,
 2753 or with calls to any other thread-safe functions, by multiple threads. Each function defined in
 2754 the System Interfaces volume of POSIX.1-2017 is thread-safe unless explicitly stated otherwise.
 2755 Examples are any “pure” function, a function which holds a mutex locked while it is accessing
 2756 static storage, or objects shared among threads.

2757 3.408 Thread-Specific Data Key

2758 A process global handle of type **pthread_key_t** which is used for naming thread-specific data.

2759 Although the same key value may be used by different threads, the values bound to the key by
 2760 *pthread_setspecific()* and accessed by *pthread_getspecific()* are maintained on a per-thread basis
 2761 and persist for the life of the calling thread.

2762 **Note:** The *pthread_getspecific()* and *pthread_setspecific()* functions are defined in detail in the System
 2763 Interfaces volume of POSIX.1-2017.

2764 3.409 Tilde Character (<tilde>)

2765 The character ' ~ '.

2766 3.410 Timeouts

2767 A method of limiting the length of time an interface will block; see also [Section 3.76](#) (on page 45).

2768 3.411 Timer

2769 A mechanism that can notify a thread when the time as measured by a particular clock has
2770 reached or passed a specified value, or when a specified amount of time has passed.

2771 3.412 Timer Overrun

2772 A condition that occurs each time a timer, for which there is already an expiration signal queued
2773 to the process, expires.

2774 3.413 Token

2775 In the shell command language, a sequence of characters that the shell considers as a single unit
2776 when reading input. A token is either an operator or a word.

2777 **Note:** The rules for reading input are defined in detail in XCU [Section 2.3](#) (on page 2347).

2778 3.414 Trace Analyzer Process

2779 A process that extracts trace events from a trace stream to retrieve information about the
2780 behavior of an application.

2781 3.415 Trace Controller Process

2782 A process that creates a trace stream for tracing a process.

2783 3.416 Trace Event

2784 A data object that represents an action executed by the system, and that is recorded in a trace
2785 stream.

2786 3.417 Trace Event Type

2787 A data object type that defines a class of trace event.

2788 3.418 Trace Event Type Mapping

2789 A one-to-one mapping between trace event types and trace event names.

2790 3.419 Trace Filter

2791 A filter that allows the trace controller process to specify those trace event types that are to be
2792 ignored; that is, not generated.

2793 3.420 Trace Generation Version

2794 A data object that is an implementation-defined character string, generated by the trace system
2795 and describing the origin and version of the trace system.

2796 3.421 Trace Log

2797 The flushed image of a trace stream, if the trace stream is created with a trace log.

2798 3.422 Trace Point

2799 An action that may cause a trace event to be generated.

2800 3.423 Trace Stream

2801 An opaque object that contains trace events plus internal data needed to interpret those trace
2802 events.

2803 3.424 Trace Stream Identifier

2804 A handle to manage tracing operations in a trace stream.

2805 3.425 Trace System

2806 A system that allows both system and user trace events to be generated into a trace stream.
2807 These trace events can be retrieved later.

2808 3.426 Traced Process

2809 A process for which at least one trace stream has been created. A traced process is also called a
2810 target process.

2811 3.427 Tracing Status of a Trace Stream

2812 A status that describes the state of an active trace stream. The tracing status of a trace stream can
2813 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:
2814 running or suspended.

2815 3.428 Typed Memory Name Space

2816 A system-wide name space that contains the names of the typed memory objects present in the
2817 system. It is configurable for a given implementation.

2818 3.429 Typed Memory Object

2819 A combination of a typed memory pool and a typed memory port. The entire contents of the
2820 pool are accessible from the port. The typed memory object is identified through a name that
2821 belongs to the typed memory name space.

2822 3.430 Typed Memory Pool

2823 An extent of memory with the same operational characteristics. Typed memory pools may be
2824 contained within each other.

2825 3.431 Typed Memory Port

2826 A hardware access path to one or more typed memory pools.

2827 3.432 Unbind

2828 Remove the association between a network address and an endpoint.

2829 3.433 Unit Data

2830 See *Datagram* in [Section 3.124](#) (on page 53).

2831 3.434 Upshifting

2832 The conversion of a lowercase character that has a single-character uppercase representation into
2833 this uppercase representation.

2834 3.435 User Database

2835 A system database that contains at least the following information for each user ID:

2836 User name

2837 Numerical user ID

2838 Initial numerical group ID

2839 Initial working directory

2840 Initial user program

2841 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under
2842 which the initial values are operative are implementation-defined.

2843 If the initial user program field is null, an implementation-defined program is used.

2844 If the initial working directory field is null, the interpretation of that field is implementation-
2845 defined.

2846 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-2017.

2847 3.436 User ID

2848 A non-negative integer that is used to identify a system user. When the identity of a user is
2849 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or
2850 a saved set-user-ID. The value (**uid_t**)-1 shall not be a valid user ID, but does have a defined use
2851 in some interfaces defined in this standard.

2852 3.437 User Name

2853 A string that is used to identify a user; see also [Section 3.435](#). To be portable across systems
2854 conforming to POSIX.1-2017, the value is composed of characters from the portable filename
2855 character set. The <hyphen-minus> character should not be used as the first character of a
2856 portable user name.

2857 3.438 User Trace Event

2858 A trace event that is generated explicitly by the application as a result of a call to
2859 *posix_trace_event()*.

2860 3.439 Utility

2861 A program, excluding special built-in utilities provided as part of the Shell Command Language,
2862 that can be called by name from a shell to perform a specific task, or related set of tasks.

2863 **Note:** For further information on special built-in utilities, see XCU [Section 2.14](#) (on page 2384).

2864 3.440 Variable

2865 In the shell command language, a named parameter.

2866 **Note:** For further information, see XCU [Section 2.5](#) (on page 2349).

2867 3.441 Vertical-Tab Character (<vertical-tab>)

2868 A character that in the output stream indicates that printing should start at the next vertical
2869 tabulation position. It is the character designated by '`\v`' in the C language. If the current
2870 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is
2871 unspecified whether this character is the exact sequence transmitted to an output device by the
2872 system to accomplish the tabulation.

2873 3.442 White Space

2874 A sequence of one or more characters that belong to the **space** character class as defined via the
2875 `LC_CTYPE` category in the current locale.

2876 In the POSIX locale, white space consists of one or more `<blank>` (`<space>` and `<tab>`
2877 characters), `<newline>`, `<carriage-return>`, `<form-feed>`, and `<vertical-tab>` characters.

2878 3.443 Wide-Character Code (C Language)

2879 An integer value corresponding to a single graphic symbol or control code.

2880 **Note:** C Language Wide-Character Codes are defined in detail in [Section 6.3](#) (on page 128).

2881 3.444 Wide-Character Input/Output Functions

2882 The functions that perform wide-oriented input from streams or wide-oriented output to
 2883 streams: *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *putwc()*,
 2884 *putwchar()*, *ungetwc()*, *vfwprintf()*, *vfwscanf()*, *vwprintf()*, *vwscanf()*, *wprintf()*, and *wscanf()*.

2885 **Note:** These functions are defined in detail in the System Interfaces volume of POSIX.1-2017.

2886 3.445 Wide-Character String

2887 A contiguous sequence of wide-character codes terminated by and including the first null wide-
 2888 character code.

2889 3.446 Word

2890 In the shell command language, a token other than an operator. In some cases a word is also a
 2891 portion of a word token: in the various forms of parameter expansion, such as *\${name-word}*,
 2892 and variable assignment, such as *name=word*, the word is the portion of the token depicted by
 2893 *word*. The concept of a word is no longer applicable following word expansions *‡*only fields
 2894 remain.

2895 **Note:** For further information, see XCU [Section 2.6.2](#) (on page 2354) and [Section 2.6](#) (on page 2353).

2896 3.447 Working Directory (or Current Working Directory)

2897 A directory, associated with a process, that is used in pathname resolution for pathnames that do
 2898 not begin with a <slash> character.

2899 3.448 Worldwide Portability Interface

2900 Functions for handling characters in a codeset-independent manner.

2901 3.449 Write

2902 To output characters to a file, such as standard output or standard error. Unless otherwise stated,
 2903 standard output is the default output destination for all uses of the term “write”; see the
 2904 distinction between display and write in [Section 3.133](#) (on page 54).

2905 3.450 XSI

2906 The X/Open System Interfaces (XSI) option is the core application programming interface for C
 2907 and *sh* programming for systems conforming to the Single UNIX Specification. This is a

2908 superset of the mandatory requirements for conformance to POSIX.1-2017.

2909 **3.451 XSI-Conformant**

2910 A system which allows an application to be built using a set of services that are consistent across
2911 all systems that conform to POSIX.1-2017 and that support the XSI option.

2912 **Note:** See also [Chapter 2](#) (on page 15).

2913 **3.452 Zombie Process**

2914 The remains of a live process (see [Section 3.210](#), on page 66) after it terminates (see [Section 3.303](#),
2915 on page 82) and before its status information (see XSH [Section 2.13](#), on page 548) is consumed by
2916 its parent process.

2917 **3.453 ± 0**

2918 The algebraic sign provides additional information about any variable that has the value zero
2919 when the representation allows the sign to be determined.

2920

2921

General Concepts

2922

For the purposes of POSIX.1-2017, the general concepts given in [Chapter 4](#) apply.

2923

Note: No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

2924

2925

2926

4.1 Concurrent Execution

2927

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

2928

2929

4.2 Default Initialization

2930

Default initialization causes an object to be initialized according to these rules:

2931

If it has pointer type, it is initialized to a null pointer.

2932

If it has arithmetic type, it is initialized to (positive or unsigned) zero.

2933

If it is an aggregate, every member is initialized (recursively) according to these rules.

2934

If it is a union, the first named member is initialized (recursively) according to these rules.

2935

For an object of aggregate type with an explicit initializer, the initialization shall occur in initializer list order, each initializer provided for a particular subobject overriding any previously listed initializer for the same subobject; all subobjects that are not initialized explicitly shall be initialized implicitly according to the rules for default initialization.

2936

2937

2938

2939

Objects with static storage duration but no explicit initializer shall be initialized implicitly according to the rules for default initialization.

2940

2941

An explicit initializer of `{ 0 }` works to perform explicit default initialization for any object of scalar or aggregate type, and for any storage duration.

2942

2943

Notes:

2944

1. The ISO C standard does not require a compiler to set any field alignment padding bits in a structure or array definition to a particular value. Because of this, a structure initialized using `{ 0 }` might not *memcmp()* as equal to the same structure initialized using *memset()* to zero. For consistent results, portable applications comparing structures should test each field individually.

2945

2946

2947

2948

2949

2. If an implementation treats the all-zero bit pattern of a pointer object as a null pointer, and the all-zero bit pattern of a floating-point object as equivalent to positive 0, then *memset()* to zero and *calloc()* have the same effects as default initialization for all named members of a structure. Implementations that define `__STDC_IEC_559__` guarantee that the all-zero bit pattern of a floating-point object represents 0.0.

2950

2951

2952

MX

2953

2954 4.3 Directory Protection

2955 If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove
2956 or rename files within that directory only if one or more of the following is true:

2957 The effective user ID of the process is the same as that of the owner ID of the file.

2958 The effective user ID of the process is the same as that of the owner ID of the directory.

2959 The process has appropriate privileges.

2960 Optionally, the file is writable by the process. Whether or not files that are writable by the
2961 process can be removed or renamed is implementation-defined.

2962 If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

2963 4.4 Extended Security Controls

2964 An implementation may provide implementation-defined extended security controls (see
2965 [Section 3.160](#), on page 58). These permit an implementation to provide security mechanisms to
2966 implement different security policies than those described in POSIX.1-2017. These mechanisms
2967 shall not alter or override the defined semantics of any of the interfaces in POSIX.1-2017.

2968 4.5 File Access Permissions

2969 The standard file access control mechanism uses the file permission bits, as described below.

2970 Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An
2971 additional access control mechanism shall only further restrict the access permissions defined by
2972 the file permission bits. An alternate file access control mechanism shall:

2973 Specify file permission bits for the file owner class, file group class, and file other class of
2974 that file, corresponding to the access permissions.

2975 Be enabled only by explicit user action, on a per-file basis by the file owner or a user with
2976 appropriate privileges.

2977 Be disabled for a file after the file permission bits are changed for that file with `chmod()`.
2978 The disabling of the alternate mechanism need not disable any additional mechanisms
2979 supported by an implementation.

2980 Whenever a process requests file access permission for read, write, or execute/search, if no
2981 additional mechanism denies access, access shall be determined as follows:

2982 If a process has appropriate privileges:

2983 if `r` or `w` or directory search permission is requested, access shall be granted.

2984 if `x` permission is requested, access shall be granted if execute permission is
2985 granted to at least one user by the file permission bits or by an alternate access
2986 control mechanism; otherwise, access shall be denied.

2987 Otherwise:

2988 if file permission bits of a file contain read, write, and execute/search permissions
2989 for the file owner class, file group class, and file other class.

2990 ‡ Access shall be granted if an alternate access control mechanism is not enabled and
 2991 the requested access permission bit is set for the class (file owner class, file group
 2992 class, or file other class) to which the process belongs, or if an alternate access control
 2993 mechanism is enabled and it allows the requested access; otherwise, access shall be
 2994 denied.

2995 4.6 File Hierarchy

2996 Files in the system are organized in a hierarchical structure in which all of the non-terminal
 2997 nodes are directories and all of the terminal nodes are any other type of file. Since multiple
 2998 directory entries may refer to the same file, the hierarchy is properly described as a “directed
 2999 graph”.

3000 4.7 Filenames

3001 Uppercase and lowercase letters shall retain their unique identities between conforming
 3002 implementations.

3003 4.8 Filename Portability

3004 For a filename to be portable across implementations conforming to POSIX.1-2017, it shall
 3005 consist only of the portable filename character set as defined in [Section 3.282](#) (on page 79).

3006 **Note:** Applications should avoid using filenames that have the <hyphen-minus> character as the first
 3007 character since this may cause problems when filenames are passed as command line
 3008 arguments.

3009 4.9 File Times Update

3010 Many operations have requirements to update file timestamps. These requirements do not apply
 3011 to streams that have no underlying file description (for example, memory streams created by
 3012 `open_memstream()` have no underlying file description).

3013 Each file has three distinct associated timestamps: the time of last data access, the time of last
 3014 data modification, and the time the file status last changed. These values are returned in the file
 3015 characteristics structure `struct stat`, as described in [<sys/stat.h>](#) (on page 392).

3016 Each function or utility in POSIX.1-2017 that reads or writes data (even if the data does not
 3017 change) or performs an operation to change file status (even if the file status does not change)
 3018 indicates which of the appropriate timestamps shall be marked for update. If an implementation
 3019 of such a function or utility marks for update one of these timestamps in a place or time not
 3020 specified by POSIX.1-2017, this shall be documented, except that any changes caused by
 3021 pathname resolution need not be documented. For the other functions or utilities in
 3022 POSIX.1-2017 (those that are not explicitly required to read or write file data or change file
 3023 status, but that in some implementations happen to do so), the effect is unspecified.

3024 An implementation may update timestamps that are marked for update immediately, or it may

3025 update such timestamps periodically. At the point in time when an update occurs, any marked
3026 timestamps shall be set to the current time and the update marks shall be cleared. All
3027 timestamps that are marked for update shall be updated when the file ceases to be open by any
3028 process or before a *fstat()*, *fstatat()*, *fsync()*, *futimens()*, *lstat()*, *stat()*, *utime()*, *utimensat()*, or
3029 *utimes()* is successfully performed on the file. Other times at which updates are done are
3030 unspecified. Marks for update, and updates themselves, shall not be done for files on read-only
3031 file systems; see [Section 3.310](#) (on page 83).

3032 The resolution of timestamps of files in a file system is implementation-defined, but shall be no
3033 coarser than one-second resolution. The three timestamps shall always have values that are
3034 supported by the file system. Whenever any of a file's timestamps are to be set to a value *V*
3035 according to the rules of the preceding paragraphs of this section, the implementation shall
3036 immediately set the timestamp to the greatest value supported by the file system that is not
3037 greater than *V*.

3038 4.10 Host and Network Byte Orders

3039 When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned
3040 values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be
3041 stored in several octets. The convention is that all such values are stored with 8 bits in each octet,
3042 and with the first (lowest-addressed) octet holding the most-significant bits. This is called
3043 "network byte order".

3044 Network byte order may not be convenient for processing actual values. For this, it is more
3045 sensible for values to be stored as ordinary integers. This is known as "host byte order". In host
3046 byte order:

3047 The most significant bit might not be stored in the first byte in address order.

3048 Bits might not be allocated to bytes in any obvious order at all.

3049 8-bit values stored in `uint8_t` objects do not require conversion to or from host byte order, as
3050 they have the same representation. 16 and 32-bit values can be converted using the *htonl()*,
3051 *htons()*, *ntohl()*, and *ntohs()* functions. When reading data that is to be converted to host byte
3052 order, it should either be received directly into a `uint16_t` or `uint32_t` object or should be copied
3053 from an array of bytes using *memcpy()* or similar. Passing the data through other types could
3054 cause the byte order to be changed. Similar considerations apply when sending data.

3055 4.11 Measurement of Execution Time

3056 The mechanism used to measure execution time shall be implementation-defined. The
3057 implementation shall also define to whom the CPU time that is consumed by interrupt handlers
3058 and system services on behalf of the operating system will be charged. See [Section 3.118](#) (on
3059 page 52).

3060 4.12 Memory Synchronization

3061 Applications shall ensure that access to any memory location by more than one thread of control
 3062 (threads or processes) is restricted such that no thread of control can read or modify a memory
 3063 location while another thread of control may be modifying it. Such access is restricted using
 3064 functions that synchronize thread execution and also synchronize memory with respect to other
 3065 threads. The following functions synchronize memory with respect to other threads:

3066	<i>fork()</i>	<i>pthread_mutex_trylock()</i>	<i>pthread_rwlock_unlock()</i>
3067	<i>pthread_barrier_wait()</i>	<i>pthread_mutex_unlock()</i>	<i>pthread_rwlock_wrlock()</i>
3068	<i>pthread_cond_broadcast()</i>	<i>pthread_spin_lock()</i>	<i>sem_post()</i>
3069	<i>pthread_cond_signal()</i>	<i>pthread_spin_trylock()</i>	<i>sem_timedwait()</i>
3070	<i>pthread_cond_timedwait()</i>	<i>pthread_spin_unlock()</i>	<i>sem_trywait()</i>
3071	<i>pthread_cond_wait()</i>	<i>pthread_rwlock_rdlock()</i>	<i>sem_wait()</i>
3072	<i>pthread_create()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>semctl()</i>
3073	<i>pthread_join()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>semop()</i>
3074	<i>pthread_mutex_lock()</i>	<i>pthread_rwlock_tryrdlock()</i>	<i>wait()</i>
3075	<i>pthread_mutex_timedlock()</i>	<i>pthread_rwlock_trywrlock()</i>	<i>waitpid()</i>

3076 The *pthread_once()* function shall synchronize memory for the first call in each thread for a given
 3077 **pthread_once_t** object. If the *init_routine* called by *pthread_once()* is a cancellation point and is
 3078 canceled, a call to *pthread_once()* for the same **pthread_once_t** object made from a cancellation
 3079 cleanup handler shall also synchronize memory.

3080 The *pthread_mutex_lock()* function need not synchronize memory if the mutex type is
 3081 PTHREAD_MUTEX_RECURSIVE and the calling thread already owns the mutex. The
 3082 *pthread_mutex_unlock()* function need not synchronize memory if the mutex type is
 3083 PTHREAD_MUTEX_RECURSIVE and the mutex has a lock count greater than one.

3084 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified
 3085 whether the invocation causes memory to be synchronized.

3086 Applications may allow more than one thread of control to read a memory location
 3087 simultaneously.

3088 4.13 Pathname Resolution

3089 Pathname resolution is performed for a process to resolve a pathname to a particular directory
 3090 entry for a file in the file hierarchy. There may be multiple pathnames that resolve to the same
 3091 directory entry, and multiple directory entries for the same file. When a process resolves a
 3092 pathname of an existing directory entry, the entire pathname shall be resolved as described
 3093 below. When a process resolves a pathname of a directory entry that is to be created immediately
 3094 after the pathname is resolved, pathname resolution terminates when all components of the path
 3095 prefix of the last component have been resolved. It is then the responsibility of the process to
 3096 create the final component.

3097 Each filename in the pathname is located in the directory specified by its predecessor (for
 3098 example, in the pathname fragment **a/b**, file **b** is located in directory **a**). Pathname resolution
 3099 shall fail if this cannot be accomplished. If the pathname begins with a <slash>, the predecessor
 3100 of the first filename in the pathname shall be taken to be the root directory of the process (such
 3101 pathnames are referred to as “absolute pathnames”). If the pathname does not begin with a
 3102 <slash>, the predecessor of the first filename of the pathname shall be taken to be either the
 3103 current working directory of the process or for certain interfaces the directory identified by a file
 3104 descriptor passed to the interface (such pathnames are referred to as “relative pathnames”).

3105 The interpretation of a pathname component is dependent on the value of {NAME_MAX} and
 3106 _POSIX_NO_TRUNC associated with the path prefix of that component. If any pathname
 3107 component is longer than {NAME_MAX}, the implementation shall consider this an error.

3108 A pathname that contains at least one non-`<slash>` character and that ends with one or more
 3109 trailing `<slash>` characters shall not be resolved successfully unless the last pathname
 3110 component before the trailing `<slash>` characters names an existing directory or a directory
 3111 entry that is to be created for a directory immediately after the pathname is resolved. Interfaces
 3112 using pathname resolution may specify additional constraints⁶ when a pathname that does not
 3113 name an existing directory contains at least one non-`<slash>` character and contains one or more
 3114 trailing `<slash>` characters.

3115 If a symbolic link is encountered during pathname resolution, the behavior shall depend on
 3116 whether the pathname component is at the end of the pathname and on the function being
 3117 performed. If all of the following are true, then pathname resolution is complete:

- 3118 1. This is the last pathname component of the pathname.
- 3119 2. The pathname has no trailing `<slash>`.
- 3120 3. The function is required to act on the symbolic link itself, or certain arguments direct that
 3121 the function act on the symbolic link itself.

3122 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the
 3123 symbolic link, except that if the contents of the symbolic link is the empty string, then either
 3124 pathname resolution shall fail with functions reporting an [ENOENT] error and utilities writing
 3125 an equivalent diagnostic message, or the pathname of the directory containing the symbolic link
 3126 shall be used in place of the contents of the symbolic link. If the contents of the symbolic link
 3127 consist solely of `<slash>` characters, then all leading `<slash>` characters of the remaining
 3128 pathname shall be omitted from the resulting combined pathname, leaving only the leading
 3129 `<slash>` characters from the symbolic link contents. In the cases where prefixing occurs, if the
 3130 combined length exceeds {PATH_MAX}, and the implementation considers this to be an error,
 3131 pathname resolution shall fail with functions reporting an [ENAMETOOLONG] error and
 3132 utilities writing an equivalent diagnostic message. Otherwise, the resolved pathname shall be
 3133 the resolution of the pathname just created. If the resulting pathname does not begin with a
 3134 `<slash>`, the predecessor of the first filename of the pathname is taken to be the directory
 3135 containing the symbolic link.

3136 If the system detects a loop in the pathname resolution process, pathname resolution shall fail
 3137 with functions reporting an [ELOOP] error and utilities writing an equivalent diagnostic
 3138 message. The same may happen if during the resolution process more symbolic links were
 3139 followed than the implementation allows. This implementation-defined limit shall not be
 3140 smaller than {SYMLOOP_MAX}.

3141 The special filename dot shall refer to the directory specified by its predecessor. The special
 3142 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,
 3143 in the root directory, dot-dot may refer to the root directory itself.

3144 A pathname consisting of a single `<slash>` shall resolve to the root directory of the process. A
 3145 null pathname shall not be successfully resolved. If a pathname begins with two successive
 3146 `<slash>` characters, the first component following the leading `<slash>` characters may be
 3147 interpreted in an implementation-defined manner, although more than two leading `<slash>`
 3148 characters shall be treated as a single `<slash>` character.

3149 Pathname resolution for a given pathname shall yield the same results when used by any

3150 6. The only interfaces that further constrain pathnames in POSIX.1-2017 are the *rename()* and *renameat()* functions (see XSH *rename()*)
 3151 and the *mv* utility (see XCU *mv*).

3152 interface in POSIX.1-2017 as long as there are no changes to any files evaluated during pathname
3153 resolution for the given pathname between resolutions.

3154 **4.14 Process ID Reuse**

3155 A process group ID shall not be reused by the system until the process group lifetime ends.

3156 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
3157 there exists a process group whose process group ID is equal to that process ID, the process ID
3158 shall not be reused by the system until the process group lifetime ends. A process that is not a
3159 system process shall not have a process ID of 1.

3160 **4.15 Scheduling Policy**

3161 A scheduling policy affects process or thread ordering:

3162 When a process or thread is a running thread and it becomes a blocked thread

3163 When a process or thread is a running thread and it becomes a preempted thread

3164 When a process or thread is a blocked thread and it becomes a runnable thread

3165 When a running thread calls a function that can change the priority or scheduling policy of
3166 a process or thread

3167 In other scheduling policy-defined circumstances

3168 Conforming implementations shall define the manner in which each of the scheduling policies
3169 may modify the priorities or otherwise affect the ordering of processes or threads at each of the
3170 occurrences listed above. Additionally, conforming implementations shall define in what other
3171 circumstances and in what manner each scheduling policy may modify the priorities or affect
3172 the ordering of processes or threads.

3173 **4.16 Seconds Since the Epoch**

3174 A value that approximates the number of seconds that have elapsed since the Epoch. A
3175 Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*),
3176 hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900
3177 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the
3178 expression below.

3179 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and
3180 the value is non-negative, the value is related to a Coordinated Universal Time name according
3181 to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all
3182 integer types:

```
3183 tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +
3184 (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 -
3185 ((tm_year-1)/100)*86400 + ((tm_year+299)/400)*86400
```

3186 The relationship between the actual time of day and the current value for seconds since the
3187 Epoch is unspecified.

3188 How any changes to the value of seconds since the Epoch are made to align to a desired
3189 relationship with the current actual time is implementation-defined. As represented in seconds
3190 since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3191 **Note:** The last three terms of the expression add in a day for each year that follows a leap year starting
3192 with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973,
3193 the second subtracts a day back out every 100 years starting in 2001, and the third adds a day
3194 back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that
3195 is, the remainder is discarded leaving only the integer quotient.

3196 4.17 Semaphore

3197 A minimum synchronization primitive to serve as a basis for more complex synchronization
3198 mechanisms to be defined by the application program.

3199 For the semaphores associated with the Semaphores option, a semaphore is represented as a
3200 shareable resource that has a non-negative integer value. When the value is zero, there is a
3201 (possibly empty) set of threads awaiting the availability of the semaphore.

3202 For the semaphores associated with the X/Open System Interfaces (XSI) option, a semaphore is
3203 a positive integer (0 through 32767). The *semget()* function can be called to create a set or array of
3204 semaphores. A semaphore set can contain one or more semaphores up to an implementation-
3205 defined value.

3206 Semaphore Lock Operation

3207 An operation that is applied to a semaphore. If, prior to the operation, the value of the
3208 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and
3209 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

3210 Semaphore Unlock Operation

3211 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in
3212 the set of threads awaiting the semaphore, then some thread from that set shall be removed from
3213 the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

3214 4.18 Thread-Safety

3215 Refer to XSH [Section 2.9](#) (on page 512).

3216 4.19 Tracing

3217 The trace system allows a traced process to have a selection of events created for it. Traces
3218 consist of streams of trace event types.

3219 A trace event type is identified on the one hand by a trace event type name, also referenced as a
3220 trace event name, and on the other hand by a trace event type identifier. A trace event name is a
3221 human-readable string. A trace event type identifier is an opaque identifier used by the trace
3222 system. There shall be a one-to-one relationship between trace event type identifiers and trace
3223 event names for a given trace stream and also for a given traced process. The trace event type
3224 identifier shall be generated automatically from a trace event name by the trace system either
3225 when a trace controller process invokes *posix_trace_trid_eventid_open()* or when an instrumented
3226 application process invokes *posix_trace_eventid_open()*. Trace event type identifiers are used to
3227 filter trace event types, to allow interpretation of user data, and to identify the kind of trace
3228 point that generated a trace event.

3229 Each trace event shall be of a particular trace event type, and associated with a trace event type
3230 identifier. The execution of a trace point shall generate a trace event if a trace stream has been
3231 created and started for the process that executed the trace point and if the corresponding trace
3232 event type identifier is not ignored by filtering.

3233 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a
3234 trace log is associated with the trace stream, except that:

3235 For a trace stream, if no resources are available for the event, the event is lost.

3236 For a trace log, if no resources are available for the event, or a flush operation does not
3237 succeed, the event is lost.

3238 A trace event recorded in an active trace stream may be retrieved by an application having
3239 appropriate privileges.

3240 A trace event recorded in a trace log may be retrieved by an application having appropriate
3241 privileges after opening the trace log as a pre-recorded trace stream, with the function
3242 *posix_trace_open()*.

3243 When a trace event is reported it is possible to retrieve the following:

3244 A trace event type identifier

3245 A timestamp

3246 The process ID of the traced process, if the trace event is process-dependent

3247 Any optional trace event data including its length

3248 If the Threads option is supported, the thread ID, if the trace event is process-dependent

3249 The program address at which the trace point was invoked

3250 Trace events may be mapped from trace event types to trace event names. One such mapping
3251 shall be associated with each trace stream. An active trace stream is associated with a traced
3252 process, and also with its children if the Trace Inherit option is supported and also the
3253 inheritance policy is set to *_POSIX_TRACE_INHERIT*. Therefore each traced process has a
3254 mapping of the trace event names to trace event type identifiers that have been defined for that
3255 process.

3256 Traces can be recorded into either trace streams or trace logs.

3257 The implementation and format of a trace stream are unspecified. A trace stream need not be
3258 and generally is not persistent. A trace stream may be either active or pre-recorded:

3259 An active trace stream is a trace stream that has been created and has not yet been shut
3260 down. It can be of one of the two following classes:

- 3261 1. An active trace stream without a trace log that was created with the
3262 *posix_trace_create()* function
- 3263 2. If the Trace Log option is supported, an active trace stream with a trace log that was
3264 created with the *posix_trace_create_withlog()* function

3265 A pre-recorded trace stream is a trace stream that was opened from a trace log object using
3266 the *posix_trace_open()* function.

3267 An active trace stream can loop. This behavior means that when the resources allocated by the
3268 trace system for the trace stream are exhausted, the trace system reuses the resources associated
3269 with the oldest recorded trace events to record new trace events.

3270 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This
3271 operation causes the trace system to write trace events from the trace stream to the associated
3272 trace log, following the defined policies or using an explicit function call. After this operation,
3273 the trace system may reuse the resources associated with the flushed trace events.

3274 An active trace stream with or without a trace log can be cleared. This operation shall cause all
3275 the resources associated with this trace stream to be reinitialized. The trace stream shall behave
3276 as if it was returning from its creation, except that the mapping of trace event type identifiers to
3277 trace event names shall not be cleared. If a trace log was associated with this trace stream, the
3278 trace log shall also be reinitialized.

3279 A trace log shall be recorded when the *posix_trace_shutdown()* operation is invoked or during
3280 tracing, depending on the tracing strategy which is defined by a log policy. After the trace
3281 stream has been shut down, the trace information can be retrieved from the associated trace log
3282 using the same interface used to retrieve information from an active trace stream.

3283 For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance
3284 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together
3285 with all of its future children. The *posix_pid* member of each trace event in a trace stream shall be
3286 the process ID of the traced process.

3287 Each trace point may be an implementation-defined action such as a context switch, or an
3288 application-programmed action such as a call to a specific operating system service (for
3289 example, *fork()*) or a call to *posix_trace_event()*.

3290 Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace
3291 events. By default, no trace events are filtered.

3292 The results of the tracing operations can be analyzed and monitored by a trace controller process
3293 or a trace analyzer process.

3294 Only the trace controller process has control of the trace stream it has created. The control of the
3295 operation of a trace stream is done using its corresponding trace stream identifier. The trace
3296 controller process is able to:

- 3297 Initialize the attributes of a trace stream
- 3298 Create the trace stream
- 3299 Start and stop tracing
- 3300 Know the mapping of the traced process

3301 If the Trace Event Filter option is supported, filter the type of trace events to be recorded

3302 Shut the trace stream down

3303 A traced process may also be a trace controller process. Only the trace controller process can
3304 control its trace stream(s). A trace stream created by a trace controller process shall be shut down
3305 if its controller process terminates or executes another file.

3306 A trace controller process may also be a trace analyzer process. Trace analysis can be done
3307 concurrently with the traced process or can be done off-line, in the same or in a different
3308 platform.

3309 4.20 Treatment of Error Conditions for Mathematical Functions

3310 For all the functions in the `<math.h>` header, an application wishing to check for error situations
3311 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On
3312 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`
3313 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3314 On implementations that support the IEC 60559 Floating-Point option, whether or when
3315 functions in the `<math.h>` header raise an undeserved underflow floating-point exception is
3316 unspecified. Otherwise, as implied by XSH `feraiseexcept()`, the `<math.h>` functions do not raise
3317 spurious floating-point exceptions (detectable by the user), other than the inexact floating-point
3318 exception.

3319 The following error conditions are defined for all functions in the `<math.h>` header.

3320 4.20.1 Domain Error

3321 A “domain error” shall occur if an input argument is outside the domain over which the
3322 mathematical function is defined. The description of each function lists any required domain
3323 errors; an implementation may define additional domain errors, provided that such errors are
3324 consistent with the mathematical definition of the function.

3325 On a domain error, the function shall return an implementation-defined value; if the integer
3326 expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to [EDOM]; if
3327 the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “invalid”
3328 floating-point exception shall be raised.

3329 4.20.2 Pole Error

3330 A “pole error” occurs if the mathematical result of the function is an exact infinity (for example,
3331 `log(0.0)`).

3332 On a pole error, the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or
3333 `HUGE_VALL` according to the return type, with the same sign as the correct value of the
3334 function; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall
3335 be set to [ERANGE]; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-
3336 zero, the “divide-by-zero” floating-point exception shall be raised.

3337 **4.20.3 Range Error**

3338 A “range error” shall occur if the finite mathematical result of the function cannot be
3339 represented in an object of the specified type, due to extreme magnitude.

3340 **4.20.3.1 Result Overflows**

3341 A floating result overflows if the magnitude of the mathematical result is finite but so large that
3342 the mathematical result cannot be represented without extraordinary roundoff error in an object
3343 of the specified type. If a floating result overflows and default rounding is in effect, then the
3344 function shall return the value of the macro HUGE_VAL, HUGE_VALF, or HUGE_VALL
3345 according to the return type, with the same sign as the correct value of the function; if the integer
3346 expression (math_errhandling & MATH_ERRNO) is non-zero, *errno* shall be set to [ERANGE]; if
3347 the integer expression (math_errhandling & MATH_ERREXCEPT) is non-zero, the “overflow”
3348 floating-point exception shall be raised.

3349 **4.20.3.2 Result Underflows**

3350 The result underflows if the magnitude of the mathematical result is so small that the
3351 mathematical result cannot be represented, without extraordinary roundoff error, in an object of
3352 the specified type. If the result underflows, the function shall return an implementation-defined
3353 value whose magnitude is no greater than the smallest normalized positive number in the
3354 specified type; if the integer expression (math_errhandling & MATH_ERRNO) is non-zero,
3355 whether *errno* is set to [ERANGE] is implementation-defined; if the integer expression
3356 (math_errhandling & MATH_ERREXCEPT) is non-zero, whether the “underflow” floating-point
3357 exception is raised is implementation-defined.

3358 **4.21 Treatment of NaN Arguments for the Mathematical Functions**

3359 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,
3360 except where stated otherwise.

3361 If a function with one or more NaN arguments returns a NaN result, the result should be the
3362 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3363 On implementations that support the IEC 60559:1989 standard floating point, functions with
3364 signaling NaN argument(s) shall be treated as if the function were called with an argument that
3365 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3366 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are
3367 evaluated during the function call.

3368 On implementations that support the IEC 60559:1989 standard floating point, for those
3369 functions that do not have a documented domain error, the following shall apply:

3370 These functions shall fail if:

3371 Domain Error Any argument is a signaling NaN.

3372 Either, the integer expression (math_errhandling & MATH_ERRNO) is non-zero and *errno*
3373 shall be set to [EDOM], or the integer expression (math_errhandling &
3374 MATH_ERREXCEPT) is non-zero and the invalid floating-point exception shall be raised.

3375 4.22 Utility

3376 A utility program shall be either an executable file, such as might be produced by a compiler or
3377 linker system from computer source code, or a file of shell source code, directly interpreted by
3378 the shell. The program may have been produced by the user, provided by the system
3379 implementor, or acquired from an independent distributor.

3380 The system may implement certain utilities as shell functions (see XCU [Section 2.9.5](#), on page
3381 2374) or built-in utilities, but only an application that is aware of the command search order (as
3382 described in XCU [Section 2.9.1.1](#), on page 2367) or of performance characteristics can discern
3383 differences between the behavior of such a function or built-in utility and that of an executable
3384 file.

3385 4.23 Variable Assignment

3386 In the shell command language, a word consisting of the following parts:

3387 *varname=value*

3388 When used in a context where assignment is defined to occur and at no other time, the *value*
3389 (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

3390 **Note:** For further information, see XCU [Section 2.9.1](#) (on page 2365).

3391 The *varname* and *value* parts shall meet the requirements for a name and a word, respectively,
3392 except that they are delimited by the embedded unquoted <equals-sign>, in addition to other
3393 delimiters.

3394 **Note:** Additional delimiters are described in XCU [Section 2.3](#) (on page 2347).

3395 When a variable assignment is done, the variable shall be created if it did not already exist. If
3396 *value* is not specified, the variable shall be given a null value.

3397 **Note:** An alternative form of variable assignment:

3398 *symbol=value*

3399 (where *symbol* is a valid word delimited by an <equals-sign>, but not a valid name) produces
3400 unspecified results. The form *symbol=value* is used by the KornShell *name[expression]=value*
3401 syntax.

File Format Notation

The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility descriptions use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces volume of POSIX.1-2017 *printf()* function, as described in this chapter. When used in STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the System Interfaces volume of POSIX.1-2017 *scanf()* function to read the input file.

The description of an individual record is as follows:

```
"<format>", [<arg1>, <arg2>, ..., <argn>]
```

The *format* is a character string that contains three types of objects defined below:

1. *Characters* that are not "escape sequences" or "conversion specifications", as described below, shall be copied to the output.
2. *Escape Sequences* represent non-graphic characters and the escape character (<backslash>).
3. *Conversion Specifications* specify the output format of each argument; see below.

The following characters have the following special meaning in the format string:

' ' (An empty character position.) Represents one or more <blank> characters.

Δ Represents exactly one <space> character.

Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

Table 5-1 Escape Sequences and Associated Actions

Escape Sequence	Represents Character	Terminal Action
\\	<backslash>	Print the <backslash> character.
\a	<alert>	Attempt to alert the user through audible or visible notification.
\b	<backspace>	Move the printing position to one column before the current position, unless the current position is the start of a line.
\f	<form-feed>	Move the printing position to the initial printing position of the next logical page.
\n	<newline>	Move the printing position to the start of the next line.
\r	<carriage-return>	Move the printing position to the start of the current line.
\t	<tab>	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
\v	<vertical-tab>	Move the printing position to the start of the next <vertical-tab> position. If there are no more <vertical-tab> positions left on the page, the behavior is undefined.

3439 Each conversion specification is introduced by the <percent-sign> character ('%'). After the
3440 character '%', the following shall appear in sequence:

3441 *flags* Zero or more *flags*, in any order, that modify the meaning of the conversion
3442 specification.

3443 *field width* An optional string of decimal digits to specify a minimum field width. For an
3444 output field, if the converted value has fewer bytes than the field width, it shall be
3445 padded on the left (or right, if the left-adjustment flag ('-'), described below, has
3446 been given) to the field width.

3447 *precision* Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion
3448 specifiers (the field is padded with leading zeros), the number of digits to appear
3449 after the radix character for the *e* and *f* conversion specifiers, the maximum
3450 number of significant digits for the *g* conversion specifier; or the maximum
3451 number of bytes to be written from a string in the *s* conversion specifier. The
3452 precision shall take the form of a <period> ('.') followed by a decimal digit
3453 string; a null digit string is treated as zero.

3454 *conversion specifier characters*
3455 A conversion specifier character (see below) that indicates the type of conversion
3456 to be applied.

3457 The *flag* characters and their meanings are:

3458 - The result of the conversion shall be left-justified within the field.

3459 + The result of a signed conversion shall always begin with a sign ('+' or '-').

3460 <space> If the first character of a signed conversion is not a sign, a <space> shall be
3461 prefixed to the result. This means that if the <space> and '+' flags both appear,
3462 the <space> flag shall be ignored.

3463 # The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s*
3464 conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it
3465 shall increase the precision to force the first digit of the result to be a zero. For *x* or
3466 *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively.
3467 For *a*, *A*, *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers, the result shall always contain a
3468 radix character, even if no digits follow the radix character. For *g* and *G* conversion
3469 specifiers, trailing zeros shall not be removed from the result as they usually are.

3470 0 For *a*, *A*, *d*, *e*, *E*, *f*, *F*, *g*, *G*, *i*, *o*, *u*, *x*, and *X* conversion specifiers, leading zeros
3471 (following any indication of sign or base) shall be used to pad to the field width
3472 rather than performing space padding, except when converting an infinity or NaN.
3473 If the '0' and '-' flags both appear, the '0' flag shall be ignored. For *d*, *i*, *o*, *u*,
3474 *x*, and *X* conversion specifiers, if a precision is specified, the '0' flag shall be
3475 ignored. For other conversion specifiers, the behavior is undefined.

3476 Each conversion specifier character shall result in fetching zero or more arguments. The results
3477 are undefined if there are insufficient arguments for the format. If the format is exhausted while
3478 arguments remain, the excess arguments shall be ignored.

3479 The conversion specifiers and their meanings are:

3480 *a,A* The floating-point number argument representing a floating-point number shall be
3481 converted in the style "[*-*]0*xh.hhhhp±d*", where there is one hexadecimal digit
3482 (which shall be non-zero if the argument is a normalized floating-point number
3483 and is otherwise unspecified) before the decimal-point character and the number
3484 of hexadecimal digits after it is equal to the precision; if the precision is missing

3485 and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact
3486 representation of the value; if the precision is missing and FLT_RADIX is not a
3487 power of 2, then the precision shall be sufficient to distinguish different floating-
3488 point values in the internal representation used by the utility, except that trailing
3489 zeros may be omitted; if the precision is zero and the # flag is not specified, no
3490 decimal-point character shall appear. The letters "abcdef" shall be used for a
3491 conversion and the letters "ABCDEF" for A conversion. The A conversion specifier
3492 produces a number with X and P instead of x and p. The exponent shall always
3493 contain at least one digit, and only as many more digits as necessary to represent
3494 the decimal exponent of 2. If the value is zero, the exponent shall be zero. A
3495 floating-point number argument representing an infinity or NaN shall be
3496 converted in the style of an f or F conversion specifier.

3497 d,i,o,u,x,X The integer argument shall be written as signed decimal (d or i), unsigned octal
3498 (o), unsigned decimal (u), or unsigned hexadecimal notation (x and X). The d and
3499 i specifiers shall convert to signed decimal in the style "[−]ddd". The x
3500 conversion specifier shall use the numbers and letters "0123456789abcdef" and
3501 the X conversion specifier shall use the numbers and letters
3502 "0123456789ABCDEF". The *precision* component of the argument shall specify
3503 the minimum number of digits to appear. If the value being converted can be
3504 represented in fewer digits than the specified minimum, it shall be expanded with
3505 leading zeros. The default precision shall be 1. The result of converting a zero
3506 value with a precision of 0 shall be no characters. If both the field width and
3507 precision are omitted, the implementation may precede, follow, or precede and
3508 follow numeric arguments of types d, i, and u with <blank> characters; arguments
3509 of type o (octal) may be preceded with leading zeros.

3510 f,F The floating-point number argument shall be written in decimal notation in the
3511 style [−]ddd.ddd, where the number of digits after the radix character (shown here
3512 as a decimal point) shall be equal to the *precision* specification. The LC_NUMERIC
3513 locale category shall determine the radix character to use in this format. If the
3514 *precision* is omitted from the argument, six digits shall be written after the radix
3515 character; if the *precision* is explicitly 0, no radix character shall appear.

3516 A floating-point number argument representing an infinity shall be converted in
3517 one of the styles "[−]inf" or "[−]infinity"; which style is implementation-
3518 defined. A floating-point number argument representing a NaN shall be converted
3519 in one of the styles "[−]nan(*n-char-sequence*)" or "[−]nan"; which style,
3520 and the meaning of any *n-char-sequence*, is implementation-defined. The F
3521 conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf",
3522 "infinity", or "nan", respectively.

3523 e,E The floating-point number argument shall be written in the style [−]d.ddde±dd (the
3524 symbol '±' indicates either a <plus-sign> or <hyphen-minus>), where there is one
3525 digit before the radix character (shown here as a decimal point) and the number of
3526 digits after it is equal to the precision. The LC_NUMERIC locale category shall
3527 determine the radix character to use in this format. When the precision is missing,
3528 six digits shall be written after the radix character; if the precision is 0, no radix
3529 character shall appear. The E conversion specifier shall produce a number with E
3530 instead of e introducing the exponent. The exponent shall always contain at least
3531 two digits. However, if the value to be written requires an exponent greater than
3532 two digits, additional exponent digits shall be written as necessary.

3533 A floating-point number argument representing an infinity or NaN shall be
3534 converted in the style of an f or F conversion specifier.

3535	g,G	The floating-point number argument shall be written in style <code>f</code> or <code>e</code> (or in style <code>F</code> or <code>E</code> in the case of a <code>G</code> conversion specifier), with the precision specifying the number of significant digits. The style used depends on the value converted: style <code>e</code> (or <code>E</code>) shall be used only if the exponent resulting from the conversion is less than <code>-4</code> or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character shall appear only if it is followed by a digit.
3536		
3537		
3538		
3539		
3540		
3541		A floating-point number argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
3542		
3543	c	The single-byte character argument shall be written.
3544	s	The argument shall be taken to be a string and bytes from the string shall be written until the end of the string or the number of bytes indicated by the <i>precision</i> specification of the argument is reached. If the precision is omitted from the argument, it shall be taken to be infinite, so all bytes up to the end of the string shall be written.
3545		
3546		
3547		
3548		
3549	%	Write a <code>'%'</code> character; no argument is converted.
3550		In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. The term “field width” should not be confused with the term “precision” used in the description of <code>%s</code> .
3551		
3552		
3553		
3554		

Examples

3554		
3555		To represent the output of a program that prints a date and time in the form Sunday, July 3,
3556		10:02, where <i>weekday</i> and <i>month</i> are strings:
3557		<code>"%s,Δ%sΔ%d,Δ%d:%.2d\n"</code> <code><weekday></code> , <code><month></code> , <code><day></code> , <code><hour></code> , <code><min></code>
3558		To show <code>'π'</code> written to 5 decimal places:
3559		<code>"piΔ=Δ%.5f\n"</code> , <code><value of π></code>
3560		To show an input file format consisting of five <code><colon></code> -separated fields:
3561		<code>"%s:%s:%s:%s:%s\n"</code> , <code><arg1></code> , <code><arg2></code> , <code><arg3></code> , <code><arg4></code> , <code><arg5></code>

Character Set

6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in [Table 6-1](#). This is used to describe characters within the text of POSIX.1-2017. The first eight entries in [Table 6-1](#) and all characters in [Table 6-2](#) (on page 130) are defined in the ISO/IEC 6429:1992 standard. The rest of the characters in [Table 6-1](#) are defined in the ISO/IEC 10646-1:2000 standard.

3571

Table 6-1 Portable Character Set

3572

3573

3574

3575

3576

3577

3578

3579

3580

3581

3582

3583

3584

3585

3586

3587

3588

3589

3590

3591

3592

3593

3594

3595

3596

3597

3598

3599

3600

3601

Symbolic Name(s)	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>, <BEL>		<U0007>	BELL
<backspace>, <BS>		<U0008>	BACKSPACE
<tab>, <HT>		<U0009>	CHARACTER TABULATION
<newline>, <LF>		<U000A>	LINE FEED (LF)
<vertical-tab>, <VT>		<U000B>	LINE TABULATION
<form-feed>, <FF>		<U000C>	FORM FEED (FF)
<carriage-return>, <CR>		<U000D>	CARRIAGE RETURN (CR)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(<U0028>	LEFT PARENTHESIS
<right-parenthesis>)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>, <hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>, <period>	.	<U002E>	FULL STOP
<slash>, <solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE
<four>	4	<U0034>	DIGIT FOUR

	Symbolic Name(s)	Glyph	UCS	Description
3602				
3603	<five>	5	<U0035>	DIGIT FIVE
3604	<six>	6	<U0036>	DIGIT SIX
3605	<seven>	7	<U0037>	DIGIT SEVEN
3606	<eight>	8	<U0038>	DIGIT EIGHT
3607	<nine>	9	<U0039>	DIGIT NINE
3608	<colon>	:	<U003A>	COLON
3609	<semicolon>	;	<U003B>	SEMICOLON
3610	<less-than-sign>	<	<U003C>	LESS-THAN SIGN
3611	<equals-sign>	=	<U003D>	EQUALS SIGN
3612	<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
3613	<question-mark>	?	<U003F>	QUESTION MARK
3614	<commercial-at>	@	<U0040>	COMMERCIAL AT
3615	<A>	A	<U0041>	LATIN CAPITAL LETTER A
3616		B	<U0042>	LATIN CAPITAL LETTER B
3617	<C>	C	<U0043>	LATIN CAPITAL LETTER C
3618	<D>	D	<U0044>	LATIN CAPITAL LETTER D
3619	<E>	E	<U0045>	LATIN CAPITAL LETTER E
3620	<F>	F	<U0046>	LATIN CAPITAL LETTER F
3621	<G>	G	<U0047>	LATIN CAPITAL LETTER G
3622	<H>	H	<U0048>	LATIN CAPITAL LETTER H
3623	<I>	I	<U0049>	LATIN CAPITAL LETTER I
3624	<J>	J	<U004A>	LATIN CAPITAL LETTER J
3625	<K>	K	<U004B>	LATIN CAPITAL LETTER K
3626	<L>	L	<U004C>	LATIN CAPITAL LETTER L
3627	<M>	M	<U004D>	LATIN CAPITAL LETTER M
3628	<N>	N	<U004E>	LATIN CAPITAL LETTER N
3629	<O>	O	<U004F>	LATIN CAPITAL LETTER O
3630	<P>	P	<U0050>	LATIN CAPITAL LETTER P
3631	<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
3632	<R>	R	<U0052>	LATIN CAPITAL LETTER R
3633	<S>	S	<U0053>	LATIN CAPITAL LETTER S
3634	<T>	T	<U0054>	LATIN CAPITAL LETTER T
3635	<U>	U	<U0055>	LATIN CAPITAL LETTER U
3636	<V>	V	<U0056>	LATIN CAPITAL LETTER V
3637	<W>	W	<U0057>	LATIN CAPITAL LETTER W
3638	<X>	X	<U0058>	LATIN CAPITAL LETTER X
3639	<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
3640	<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
3641	<left-square-bracket>	[<U005B>	LEFT SQUARE BRACKET
3642	<backslash>, <reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
3643	<right-square-bracket>]	<U005D>	RIGHT SQUARE BRACKET
3644	<circumflex-accent>, <circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
3645	<low-line>, <underscore>	_	<U005F>	LOW LINE
3646	<grave-accent>	`	<U0060>	GRAVE ACCENT
3647	<a>	a	<U0061>	LATIN SMALL LETTER A
3648		b	<U0062>	LATIN SMALL LETTER B
3649	<c>	c	<U0063>	LATIN SMALL LETTER C
3650	<d>	d	<U0064>	LATIN SMALL LETTER D
3651	<e>	e	<U0065>	LATIN SMALL LETTER E
3652	<f>	f	<U0066>	LATIN SMALL LETTER F
3653	<g>	g	<U0067>	LATIN SMALL LETTER G

	Symbolic Name(s)	Glyph	UCS	Description
3654				
3655	<h>	h	<U0068>	LATIN SMALL LETTER H
3656	<i>	i	<U0069>	LATIN SMALL LETTER I
3657	<j>	j	<U006A>	LATIN SMALL LETTER J
3658	<k>	k	<U006B>	LATIN SMALL LETTER K
3659	<l>	l	<U006C>	LATIN SMALL LETTER L
3660	<m>	m	<U006D>	LATIN SMALL LETTER M
3661	<n>	n	<U006E>	LATIN SMALL LETTER N
3662	<o>	o	<U006F>	LATIN SMALL LETTER O
3663	<p>	p	<U0070>	LATIN SMALL LETTER P
3664	<q>	q	<U0071>	LATIN SMALL LETTER Q
3665	<r>	r	<U0072>	LATIN SMALL LETTER R
3666	<s>	s	<U0073>	LATIN SMALL LETTER S
3667	<t>	t	<U0074>	LATIN SMALL LETTER T
3668	<u>	u	<U0075>	LATIN SMALL LETTER U
3669	<v>	v	<U0076>	LATIN SMALL LETTER V
3670	<w>	w	<U0077>	LATIN SMALL LETTER W
3671	<x>	x	<U0078>	LATIN SMALL LETTER X
3672	<y>	y	<U0079>	LATIN SMALL LETTER Y
3673	<z>	z	<U007A>	LATIN SMALL LETTER Z
3674	<left-brace>, <left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
3675	<vertical-line>		<U007C>	VERTICAL LINE
3676	<right-brace>, <right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
3677	<tilde>	~	<U007E>	TILDE

3678 POSIX.1-2017 uses character names other than the above, but only in an informative way; for
 3679 example, in examples to illustrate the use of characters beyond the portable character set with
 3680 the facilities of POSIX.1-2017.

3681 [Table 6-1](#) (on page 125) defines the characters in the portable character set and the corresponding
 3682 symbolic character names used to identify each character in a character set description file.
 3683 Characters defined in [Table 6-2](#) (on page 130) may also be used in character set description files.

3684 POSIX.1-2017 places only the following requirements on the encoded values of the characters in
 3685 the portable character set:

3686 If the encoded values associated with each member of the portable character set are not
 3687 invariant across all locales supported by the implementation, if an application uses any
 3688 pair of locales where the character encodings differ, or accesses data from an application
 3689 using a locale which has different encodings from the locales used by the application, the
 3690 results are unspecified.

3691 The encoded values associated with the digits 0 to 9 shall be such that the value of each
 3692 character after 0 shall be one greater than the value of the previous character.

3693 A null character, NUL, which has all bits set to zero, shall be in the set of characters.

3694 The encoded values associated with <period>, <slash>, <newline>, and <carriage-return>
 3695 shall be invariant across all locales supported by the implementation.

3696 The encoded values associated with the members of the portable character set are each
 3697 represented in a single byte. Moreover, if the value is stored in an object of C-language
 3698 type **char**, it is guaranteed to be positive (except the NUL, which is always zero).

3699 Conforming implementations shall support certain character and character set attributes, as
 3700 defined in [Section 7.2](#) (on page 136).

3701 6.2 Character Encoding

3702 The POSIX locale shall contain 256 single-byte characters including the characters in [Table 6-1](#)
3703 (on page 125) and [Table 6-2](#) (on page 130), which have the properties listed in [Section 7.3.1](#) (on
3704 page 139). It is unspecified whether characters not listed in those two tables are classified as
3705 **punct** or **cntrl**, or neither. Other locales shall contain the characters in [Table 6-1](#) (on page 125)
3706 and may contain any or all of the control characters identified in [Table 6-2](#) (on page 130); the
3707 presence, meaning, and representation of any additional characters are locale-specific.

3708 In locales other than the POSIX locale, a character may have a state-dependent encoding. There
3709 are two types of these encodings:

3710 A single-shift encoding (where each character not in the initial shift state is preceded by a
3711 shift code) can be defined if each shift-code and character sequence is considered a multi-
3712 byte character. This is done using the concatenated-constant format in a character set
3713 description file, as described in [Section 6.4](#) (on page 129). If the implementation supports a
3714 character encoding of this type, all of the standard utilities in the Shell and Utilities volume
3715 of POSIX.1-2017 shall support it. Use of a single-shift encoding with any of the functions in
3716 the System Interfaces volume of POSIX.1-2017 that do not specifically mention the effects
3717 of state-dependent encoding is implementation-defined.

3718 A locking-shift encoding (where the state of the character is determined by a shift code
3719 that may affect more than the single character following it) cannot be defined with the
3720 current character set description file format. Use of a locking-shift encoding with any of
3721 the standard utilities in the Shell and Utilities volume of POSIX.1-2017 or with any of the
3722 functions in the System Interfaces volume of POSIX.1-2017 that do not specifically mention
3723 the effects of state-dependent encoding is implementation-defined.

3724 While in the initial shift state, all characters in the portable character set shall retain their usual
3725 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the
3726 sequence shall be a function of the current shift state. A byte with all bits zero shall be
3727 interpreted as the null character independent of shift state. Such a byte shall not occur as part of
3728 any other character. Likewise, the byte values used to encode <period>, <slash>, <newline>, and
3729 <carriage-return> shall not occur as part of any other character in any locale.

3730 The maximum allowable number of bytes in a character in the current locale shall be indicated
3731 by {MB_CUR_MAX}, defined in the [<stdlib.h>](#) header and by the [<mb_cur_max>](#) value in a
3732 character set description file; see [Section 6.4](#) (on page 129). The implementation's maximum
3733 number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}.

3734 6.3 C Language Wide-Character Codes

3735 In the shell, the standard utilities are written so that the encodings of characters are described by
3736 the locale's *LC_CTYPE* definition (see [Section 7.3.1](#), on page 139) and there is no differentiation
3737 between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C
3738 language, a differentiation is made. To ease the handling of variable length characters, the C
3739 language has introduced the concept of wide-character codes.

3740 All wide-character codes in a given process consist of an equal number of bits. This is in contrast
3741 to characters, which can consist of a variable number of bytes. The byte or byte sequence that
3742 represents a character can also be represented as a wide-character code. Wide-character codes
3743 thus provide a uniform size for manipulating text data. A wide-character code having all bits
3744 zero is the null wide-character code (see [Section 3.251](#), on page 73), and terminates wide-
3745 character strings (see [Section 3.443](#), on page 104). The wide-character value for each member of

3746 the portable character set shall equal its value when used as the lone character in an integer
3747 character constant. Wide-character codes for other characters are locale and implementation-
3748 defined. State shift bytes shall not have a wide-character code representation. POSIX.1-2017
3749 provides no means of defining a wide-character codeset.

3750 6.4 Character Set Description File

3751 Implementations shall provide a character set description file for at least one coded character set
3752 supported by the implementation. These files are referred to elsewhere in POSIX.1-2017 as
3753 *charmap* files. It is implementation-defined whether or not users or applications can provide
3754 additional character set description files.

3755 POSIX.1-2017 does not require that multiple character sets or codesets be supported. Although
3756 multiple charmap files are supported, it is the responsibility of the implementation to provide
3757 the file or files; if only one is provided, only that one is accessible using the *localedef* utility's *-f*
3758 option.

3759 Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard
3760 position values as the encoding values, shall define characteristics for the coded character set
3761 and the encoding for the characters specified in [Table 6-1](#) (on page 125), and may define
3762 encoding for additional characters supported by the implementation. Other information about
3763 the coded character set may also be in the file. Coded character set character values shall be
3764 defined using symbolic character names followed by character encoding values.

3765 Each symbolic name specified in [Table 6-1](#) (on page 125) shall be included in the file. Each
3766 character in [Table 6-1](#) (on page 125) (each row in the table) shall be mapped to a unique coding
3767 value. For each character in [Table 6-2](#) (on page 130) that exists in the character set described by
3768 the file, the character's symbolic name(s) from [Table 6-2](#) (on page 130) and the character's single-
3769 byte encoding value shall be included in the file.

3770

Table 6-2 Non-Portable Control Characters

Symbolic Name(s)	UCS	Description
<SOH>	<U0001>	START OF HEADING
<STX>	<U0002>	START OF TEXT
<ETX>	<U0003>	END OF TEXT
<EOT>	<U0004>	END OF TRANSMISSION
<ENQ>	<U0005>	ENQUIRY
<ACK>	<U0006>	ACKNOWLEDGE
<SO>	<U000E>	SHIFT OUT
<SI>	<U000F>	SHIFT IN
<DLE>	<U0010>	DATA LINK ESCAPE
<DC1>	<U0011>	DEVICE CONTROL ONE
<DC2>	<U0012>	DEVICE CONTROL TWO
<DC3>	<U0013>	DEVICE CONTROL THREE
<DC4>	<U0014>	DEVICE CONTROL FOUR
<NAK>	<U0015>	NEGATIVE ACKNOWLEDGE
<SYN>	<U0016>	SYNCHRONOUS IDLE
<ETB>	<U0017>	END OF TRANSMISSION BLOCK
<CAN>	<U0018>	CANCEL
	<U0019>	END OF MEDIUM
<SUB>	<U001A>	SUBSTITUTE
<ESC>	<U001B>	ESCAPE
<IS4>, <FS>	<U001C>	INFORMATION SEPARATOR FOUR
<IS3>, <GS>	<U001D>	INFORMATION SEPARATOR THREE
<IS2>, <RS>	<U001E>	INFORMATION SEPARATOR TWO
<IS1>, <US>	<U001F>	INFORMATION SEPARATOR ONE
	<U007F>	DELETE

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more <blank> characters, followed by the value to be assigned to the symbol.

<code_set_name> The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with visible glyphs defined in Table 6-1 (on page 125).

<mb_cur_max> The maximum number of bytes in a multi-byte character. This shall default to 1.

<mb_cur_min> An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. On XSI-conformant systems, <mb_cur_min> shall always be 1.

<escape_char> The character used to indicate that the characters following shall be interpreted in a special way, as defined later in this section. This shall default to <backslash> ('\\'), which is the character used in all the following text and examples, unless otherwise noted.

<comment_char> The character that, when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the <number-sign> ('#').

The character set mapping definitions shall be all the lines immediately following an identifier

3817 line containing the string "CHARMAP" starting in column 1, and preceding a trailer line
 3818 containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a
 3819 **<comment_char>** in the first column shall be ignored. Each non-comment line of the character
 3820 set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file)
 3821 shall be in either of two forms:

3822 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>

3823 or:

3824 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,
 3825 <encoding>, <comments>

3826 In the first format, the line in the character set mapping definition shall define a single symbolic
 3827 name and a corresponding encoding. A symbolic name is one or more characters from the set
 3828 shown with visible glyphs in [Table 6-1](#) (on page 125), enclosed between angle brackets. A
 3829 character following an escape character is interpreted as itself; for example, the sequence
 3830 "<\\>" represents the symbolic name "\" enclosed between angle brackets.

3831 In the second format, the line in the character set mapping definition shall define a range of one
 3832 or more symbolic names. In this form, the symbolic names shall consist of zero or more non-
 3833 numeric characters from the set shown with visible glyphs in [Table 6-1](#) (on page 125), followed
 3834 by an integer formed by one or more decimal digits. Both integers shall contain the same
 3835 number of digits. The characters preceding the integer shall be identical in the two symbolic
 3836 names, and the integer formed by the digits in the second symbolic name shall be equal to or
 3837 greater than the integer formed by the digits in the first name. This shall be interpreted as a
 3838 series of symbolic names formed from the common part and each of the integers between the
 3839 first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the
 3840 symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3841 A character set mapping definition line shall exist for all symbolic names specified in [Table 6-1](#)
 3842 (on page 125), and shall define the coded character value that corresponds to the character
 3843 indicated in the table, or the coded character value that corresponds to the control character
 3844 symbolic name. If the control characters commonly associated with the symbolic names in [Table](#)
 3845 [6-2](#) (on page 130) are supported by the implementation, the symbolic name and the
 3846 corresponding encoding value shall be included in the file. Additional unique symbolic names
 3847 may be included. A coded character value can be represented by more than one symbolic name.

3848 The encoding part is expressed as one (for single-byte character values) or more concatenated
 3849 decimal, octal, or hexadecimal constants in the following formats:

3850 "%cd%u", <escape_char>, <decimal byte value>

3851 "%cx%x", <escape_char>, <hexadecimal byte value>

3852 "%c%o", <escape_char>, <octal byte value>

3853 Decimal constants shall be represented by two or three decimal digits, preceded by the escape
 3854 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".
 3855 Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape
 3856 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal
 3857 constants shall be represented by two or three octal digits, preceded by the escape character; for
 3858 example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an
 3859 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the
 3860 same type, and interpreted in sequence from first to last with the first byte of the multi-
 3861 byte character specified by the first byte in the sequence. The manner in which these constants
 3862 are represented in the character stored in the system is implementation-defined. (This notation
 3863 was chosen for reasons of portability. There is no requirement that the internal representation in
 3864 the computer memory be in this same order.) Omitting bytes from a multi-byte character

3865 definition produces undefined results.

3866 In lines defining ranges of symbolic names, the encoded value shall be the value for the first
 3867 symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic
 3868 names defined by the range shall have encoding values in increasing order. Bytes shall be
 3869 treated as unsigned octets, and carry shall be propagated between the bytes as necessary to
 3870 represent the range. However, because this causes a null byte in the second or subsequent bytes
 3871 of a character, such a declaration should not be specified. For example, the line:

```
3872 <j0101>...<j0104> \d129\d254
```

3873 is interpreted as:

```
3874 <j0101>          \d129\d254
3875 <j0102>          \d129\d255
3876 <j0103>          \d130\d00
3877 <j0104>          \d130\d01
```

3878 The expanded declaration of the symbol <j0103> in the above example is an invalid
 3879 specification, because it contains a null byte in the second byte of a character.

3880 The comment is optional.

3881 POSIX.1-2017 provides no means of defining a wide-character codeset.

3882 The following declarations can follow the character set mapping definitions (after the "END
 3883 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in
 3884 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3885 **WIDTH** A non-negative integer value defining the column width (see [Section 3.103](#), on
 3886 page 50) for the printable characters in the coded character set specified in [Table 6-1](#)
 3887 (on page 125) and [Table 6-2](#) (on page 130). Coded character set character values
 3888 shall be defined using symbolic character names followed by column width
 3889 values. Defining a character with more than one **WIDTH** produces undefined
 3890 results. The **END WIDTH** keyword shall be used to terminate the **WIDTH**
 3891 definitions. Specifying the width of a non-printable character in a **WIDTH**
 3892 declaration produces undefined results.

3893 **WIDTH_DEFAULT**

3894 A non-negative integer value defining the default column width for any printable
 3895 character not listed by one of the **WIDTH** keywords. If no **WIDTH_DEFAULT**
 3896 keyword is included in the charmap, the default character width shall be 1.

3897 **Example**

3898 After the "END CHARMAP" statement, a syntax for a width definition would be:

```
3899 WIDTH
3900 <A> 1
3901 <B> 1
3902 <C>...<Z> 1
3903 ...
3904 <fool>...<foon> 2
3905 ...
3906 END WIDTH
```

3907 In this example, the numerical code point values represented by the symbols <A> and are
 3908 assigned a width of 1. The code point values <C> to <Z> inclusive (<C>, <D>, <E>, and so on)
 3909 are also assigned a width of 1. Using <A>...<Z> would have required fewer lines, but the

3910 alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have
3911 been added as appropriate.

3912 **6.4.1 State-Dependent Character Encodings**

3913 This section addresses the use of state-dependent character encodings (that is, those in which the
3914 encoding of a character is dependent on one or more shift codes that may precede it).

3915 A single-shift encoding (where each character not in the initial shift state is preceded by a shift
3916 code) can be defined in the charmap format if each shift-code/character sequence is considered
3917 a multi-byte character, defined using the concatenated-constant format described in [Section 6.4](#)
3918 (on page 129). If the implementation supports a character encoding of this type, all of the
3919 standard utilities shall support it. A locking-shift encoding (where the state of the character is
3920 determined by a shift code that may affect more than the single character following it) could be
3921 defined with an extension to the charmap format described in [Section 6.4](#) (on page 129).

3922 If the implementation supports a character encoding of this type, any of the standard utilities
3923 that describe character (*versus* byte) or text-file manipulation shall have the following
3924 characteristics:

- 3925 1. The utility shall process the statefully encoded data as a concatenation of state-
3926 independent characters. The presence of redundant locking shifts shall not affect the
3927 comparison of two statefully encoded strings.
- 3928 2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall
3929 produce output that contains locking shifts at the beginning or end of the resulting data,
3930 if appropriate, to retain correct state information.

Locale

3933 7.1 General

3934 A locale is the definition of the subset of a user's environment that depends on language and
 3935 cultural conventions. It is made up from one or more categories. Each category is identified by
 3936 its name and controls specific aspects of the behavior of components of the system. Category
 3937 names correspond to the following environment variable names:

3938 *LC_CTYPE* Character classification and case conversion.

3939 *LC_COLLATE* Collation order.

3940 *LC_MONETARY* Monetary formatting.

3941 *LC_NUMERIC* Numeric, non-monetary formatting.

3942 *LC_TIME* Date and time formats.

3943 *LC_MESSAGES* Formats of informative and diagnostic messages and interactive responses.

3944 The standard utilities in the Shell and Utilities volume of POSIX.1-2017 shall base their behavior
 3945 on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility.
 3946 The behavior of some of the C-language functions defined in the System Interfaces volume of
 3947 POSIX.1-2017 shall also be modified based on a locale selection. The locale to be used by these
 3948 functions can be selected in the following ways:

- 3949 1. For functions such as *isalnum_l()* that take a locale object as an argument, a locale object
 3950 can be obtained from *newlocale()* or *duplocale()* and passed to the function.
- 3951 2. For functions that do not take a locale object as an argument, the current locale for the
 3952 thread can be set by calling *uselocale()* or the global locale for the process can be set by
 3953 calling *setlocale()*. Such functions shall use the current locale of the calling thread if one
 3954 has been set for that thread; otherwise, they shall use the global locale.

3955 Locales other than those supplied by the implementation can be created via the *localedef* utility,
 3956 provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is
 3957 not provided, all implementations conforming to the System Interfaces volume of POSIX.1-2017
 3958 shall provide one or more locales that behave as described in this chapter. The input to the
 3959 utility is described in [Section 7.3](#) (on page 136). The value that is used to specify a locale when
 3960 using environment variables shall be the string specified as the *name* operand to the *localedef*
 3961 utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for
 3962 the POSIX locale (see [Section 7.2](#), on page 136). When the value of a locale environment variable
 3963 begins with a <slash> (' / '), it shall be interpreted as the pathname of the locale definition; the
 3964 type of file (regular, directory, and so on) used to store the locale definition is implementation-
 3965 defined. If the value does not begin with a <slash>, the mechanism used to locate the locale is
 3966 implementation-defined.

3967 If different character sets are used by the locale categories, the results achieved by an application
 3968 utilizing these categories are undefined. Likewise, if different codesets are used for the data

3969 being processed by interfaces whose behavior is dependent on the current locale, or the codeset
3970 is different from the codeset assumed when the locale was created, the result is also undefined.

3971 Applications can select the desired locale by calling the *newlocale()* or *setlocale()* function with
3972 the appropriate value. If the function is invoked with an empty string, such as:

```
3973 newlocale(LC_ALL_MASK, "", (locale_t)0);
```

3974 or:

```
3975 setlocale(LC_ALL, "");
```

3976 the value of the corresponding environment variable is used. If the environment variable is
3977 unset or is set to the empty string, the implementation shall set the appropriate environment as
3978 defined in [Chapter 8](#) (on page 173).

3979 7.2 POSIX Locale

3980 Conforming systems shall provide a POSIX locale, also known as the C locale. In POSIX.1 the
3981 requirements for the POSIX locale are more extensive than the requirements for the C locale as
3982 specified in the ISO C standard. However, in a conforming POSIX implementation, the POSIX
3983 locale and the C locale are identical. The behavior of standard utilities and functions in the
3984 POSIX locale shall be as if the locale was defined via the *localedef* utility with input data from the
3985 POSIX locale tables in [Section 7.3](#).

3986 For C-language programs, the POSIX locale shall be the default locale when the *setlocale()*
3987 function is not called.

3988 The POSIX locale can be specified by assigning to the appropriate environment variables the
3989 values "C" or "POSIX".

3990 All implementations shall define a locale as the default locale, to be invoked when no
3991 environment variables are set, or set to the empty string. This default locale can be the POSIX
3992 locale or any other implementation-defined locale. Some implementations may provide facilities
3993 for local installation administrators to set the default locale, customizing it for each location.
3994 POSIX.1-2017 does not require such a facility.

3995 7.3 Locale Definition

3996 The capability to specify additional locales to those provided by an implementation is optional,
3997 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only
3998 implementation-supplied locales are available. Such locales shall be documented using the
3999 format specified in this section.

4000 Locales can be described with the file format presented in this section. The file format is that
4001 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the
4002 "locale definition file", but no locales shall be affected by this file unless it is processed by
4003 *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility
4004 shall apply to *localedef* or to any other similar utility used to install locale information using the
4005 locale definition file format described here.

4006 The locale definition file shall contain one or more locale category source definitions, and shall
4007 not contain more than one definition for the same locale category. If the file contains source
4008 definitions for more than one category, implementation-defined categories, if present, shall

4009 appear after the categories defined by [Section 7.1](#) (on page 135). A category source definition
 4010 contains either the definition of a category or a **copy** directive. For a description of the **copy**
 4011 directive, see *localedef*. In the event that some of the information for a locale category, as
 4012 specified in this volume of POSIX.1-2017, is missing from the locale source definition, the
 4013 behavior of that category, if it is referenced, is unspecified.

4014 A category source definition shall consist of a category header, a category body, and a category
 4015 trailer. A category header shall consist of the character string naming of the category, beginning
 4016 with the characters *LC_*. The category trailer shall consist of the string "END", followed by one
 4017 or more <blank> characters and the string used in the corresponding category header.

4018 The category body shall consist of one or more lines of text. Each line shall contain an identifier,
 4019 optionally followed by one or more operands. Identifiers shall be either keywords, identifying a
 4020 particular locale element, or collating elements. In addition to the keywords defined in this
 4021 volume of POSIX.1-2017, the source can contain implementation-defined keywords. Each
 4022 keyword within a locale shall have a unique name (that is, two categories cannot have a
 4023 commonly-named keyword); no keyword shall start with the characters *LC_*. Identifiers shall be
 4024 separated from the operands by one or more <blank> characters.

4025 Operands shall be characters, collating elements, or strings of characters. Strings shall be
 4026 enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <escape
 4027 character>, described below. When a keyword is followed by more than one operand, the
 4028 operands shall be separated by <semicolon> characters; <blank> characters shall be allowed
 4029 both before and after a <semicolon>.

4030 The first category header in the file can be preceded by a line modifying the comment character.
 4031 It shall have the following format, starting in column 1:

```
4032 "comment_char %c\n", <comment character>
```

4033 The comment character shall default to the <number-sign> ('#'). Blank lines and lines
 4034 containing the <comment character> in the first position shall be ignored.

4035 The first category header in the file can be preceded by a line modifying the escape character to
 4036 be used in the file. It shall have the following format, starting in column 1:

```
4037 "escape_char %c\n", <escape character>
```

4038 The escape character shall default to <backslash>, which is the character used in all examples
 4039 shown in this volume of POSIX.1-2017.

4040 A line can be continued by placing an escape character as the last character on the line; this
 4041 continuation character shall be discarded from the input. Although the implementation need not
 4042 accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall
 4043 place no limits on the accumulated length of the continued line. Comment lines shall not be
 4044 continued on a subsequent line using an escaped <newline>.

4045 Individual characters, characters in strings, and collating elements shall be represented using
 4046 symbolic names, as defined below. In addition, characters can be represented using the
 4047 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic
 4048 notation is used, the resultant locale definitions are in many cases not portable between systems.
 4049 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when
 4050 used to represent itself it shall be preceded by the escape character. The following rules apply to
 4051 character representation:

- 4052 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'
 4053 and '>'. The symbolic name, including the angle brackets, shall exactly match a
 4054 symbolic name defined in the charmap file specified via the *localedef* -f option, and it shall
 4055 be replaced by a character value determined from the value associated with the symbolic

4056 name in the charmap file. The use of a symbolic name not found in the charmap file shall
 4057 constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it
 4058 shall constitute a warning condition (see *localedef* for a description of actions resulting
 4059 from errors and warnings). The specification of a symbolic name in a **collating-element**
 4060 or **collating-symbol** section that duplicates a symbolic name in the charmap file (if
 4061 present) shall be an error. Use of the escape character or a right angle bracket within a
 4062 symbolic name is invalid unless the character is preceded by the escape character.

4063 For example:

4064 `<c>;<c-cedilla> "<M><a><y>"`

4065 2. A character in the portable character set can be represented by the character itself, in
 4066 which case the value of the character is implementation-defined. (Implementations may
 4067 allow other characters to be represented as themselves, but such locale definitions are not
 4068 portable.) Within a string, the double-quote character, the escape character, and the right
 4069 angle bracket character shall be escaped (preceded by the escape character) to be
 4070 interpreted as the character itself. Outside strings, the characters:

4071 `, ; < > escape_char`

4072 shall be escaped to be interpreted as the character itself.

4073 For example:

4074 `c "May"`

4075 3. A character can be represented as an octal constant. An octal constant shall be specified as
 4076 the escape character followed by two or three octal digits. Each constant shall represent a
 4077 byte value. Multi-byte values can be represented by concatenated constants specified in
 4078 byte order with the last constant specifying the least significant byte of the character.

4079 For example:

4080 `\143;\347;\143\150 "\115\141\171"`

4081 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall
 4082 be specified as the escape character followed by an 'x' followed by two hexadecimal
 4083 digits. Each constant shall represent a byte value. Multi-byte values can be represented by
 4084 concatenated constants specified in byte order with the last constant specifying the least
 4085 significant byte of the character.

4086 For example:

4087 `\x63;\xe7;\x63\x68 "\x4d\x61\x79"`

4088 5. A character can be represented as a decimal constant. A decimal constant shall be
 4089 specified as the escape character followed by a 'd' followed by two or three decimal
 4090 digits. Each constant represents a byte value. Multi-byte values can be represented by
 4091 concatenated constants specified in byte order with the last constant specifying the least
 4092 significant byte of the character.

4093 For example:

4094 `\d99;\d231;\d99\d104 "\d77\d97\d121"`

4095 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the
 4096 escape character. Only characters existing in the character set for which the locale definition is
 4097 created shall be specified, whether using symbolic names, the characters themselves, or octal,
 4098 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the
 4099 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not

4100 present in the charmap file can be specified and shall be ignored, as specified under item 1
4101 above.

4102 7.3.1 LC_CTYPE

4103 The *LC_CTYPE* category shall define character classification, case conversion, and other
4104 character attributes. In addition, a series of characters can be represented by three adjacent
4105 <period> characters representing an ellipsis symbol ("..."). The ellipsis specification shall be
4106 interpreted as meaning that all values between the values preceding and following it represent
4107 valid characters. The ellipsis specification shall be valid only within a single encoded character
4108 set; that is, within a group of characters of the same size. An ellipsis shall be interpreted as
4109 including in the list all characters with an encoded value higher than the encoded value of the
4110 character preceding the ellipsis and lower than the encoded value of the character following the
4111 ellipsis.

4112 For example:

```
4113 \x30; ... ; \x39;
```

4114 includes in the character class all characters with encoded values between the endpoints.

4115 The following keywords shall be recognized. In the descriptions, the term “automatically
4116 included” means that it shall not be an error either to include or omit any of the referenced
4117 characters; the implementation provides them if missing (even if the entire keyword is missing)
4118 and accepts them silently if present. When the implementation automatically includes a missing
4119 character, it shall have an encoded value dependent on the charmap file in effect (see the
4120 description of the *localedef -f* option); otherwise, it shall have a value derived from an
4121 implementation-defined character mapping.

4122 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included
4123 characters. These only need to be specified if the character values (that is, encoding) differ from
4124 the implementation default values. It is not possible to define a locale without these
4125 automatically included characters unless some implementation extension is used to prevent
4126 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and,
4127 thus, it might not be possible for conforming applications to work properly.

4128 **copy** Specify the name of an existing locale which shall be used as the definition of
4129 this category. If this keyword is specified, no other keyword shall be specified.

4130 **upper** Define characters to be classified as uppercase letters.

4131 In the POSIX locale, only:

```
4132 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

4133 shall be included:

4134 In a locale definition file, no character specified for the keywords **ctrl**, **digit**,
4135 **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as
4136 defined in [Section 6.4](#) (on page 129) (the portable character set), are
4137 automatically included in this class.

4138 **lower** Define characters to be classified as lowercase letters.

4139 In the POSIX locale, only:

```
4140 a b c d e f g h i j k l m n o p q r s t u v w x y z
```

4141 shall be included.

4142		In a locale definition file, no character specified for the keywords cntrl , digit ,
4143		punct , or space shall be specified. The lowercase letters <a> to <z> of the
4144		portable character set are automatically included in this class.
4145	alpha	Define characters to be classified as letters.
4146		In the POSIX locale, only characters in the classes upper and lower shall be
4147		included.
4148		In a locale definition file, no character specified for the keywords cntrl , digit ,
4149		punct , or space shall be specified. Characters classified as either upper or
4150		lower are automatically included in this class.
4151	digit	Define the characters to be classified as numeric digits.
4152		In the POSIX locale, only:
4153		0 1 2 3 4 5 6 7 8 9
4154		shall be included.
4155		In a locale definition file, only the digits <zero>, <one>, <two>, <three>,
4156		<four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
4157		contiguous ascending sequence by numerical value. The digits <zero> to
4158		<nine> of the portable character set are automatically included in this class.
4159	alnum	Define characters to be classified as letters and numeric digits. Only the
4160		characters specified for the alpha and digit keywords shall be specified.
4161		Characters specified for the keywords alpha and digit are automatically
4162		included in this class.
4163	space	Define characters to be classified as white-space characters.
4164		In the POSIX locale, exactly <space>, <form-feed>, <newline>, <carriage-
4165		return>, <tab>, and <vertical-tab> shall be included.
4166		In a locale definition file, no character specified for the keywords upper ,
4167		lower , alpha , digit , graph , or xdigit shall be specified. The <space>, <form-
4168		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
4169		character set, and any characters included in the class blank are automatically
4170		included in this class.
4171	cntrl	Define characters to be classified as control characters.
4172		In the POSIX locale, no characters in classes alpha or print shall be included.
4173		In a locale definition file, no character specified for the keywords upper ,
4174		lower , alpha , digit , punct , graph , print , or xdigit shall be specified.
4175	punct	Define characters to be classified as punctuation characters.
4176		In the POSIX locale, neither the <space> nor any characters in classes alpha ,
4177		digit , or cntrl shall be included.
4178		In a locale definition file, no character specified for the keywords upper ,
4179		lower , alpha , digit , cntrl , xdigit , or as the <space> shall be specified.
4180	graph	Define characters to be classified as printable characters, not including the
4181		<space>.
4182		In the POSIX locale, all characters in classes alpha , digit , and punct shall be
4183		included; no characters in class cntrl shall be included.

4184 In a locale definition file, characters specified for the keywords **upper**, **lower**,
 4185 **alpha**, **digit**, **xdigit**, and **punct** are automatically included in this class. No
 4186 character specified for the keyword **cntrl** shall be specified.

4187 **print** Define characters to be classified as printable characters, including the
 4188 <space>.

4189 In the POSIX locale, all characters in class **graph** shall be included; no
 4190 characters in class **cntrl** shall be included.

4191 In a locale definition file, characters specified for the keywords **upper**, **lower**,
 4192 **alpha**, **digit**, **xdigit**, **punct**, **graph**, and the <space> are automatically included
 4193 in this class. No character specified for the keyword **cntrl** shall be specified.

4194 **xdigit** Define the characters to be classified as hexadecimal digits.
 4195 In the POSIX locale, only:
 4196 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
 4197 shall be included.

4198 In a locale definition file, only the characters defined for the class **digit** shall be
 4199 specified, in contiguous ascending sequence by numerical value, followed by
 4200 one or more sets of six characters representing the hexadecimal digits 10 to 15
 4201 inclusive, with each set in ascending order (for example, <A>, , <C>, <D>,
 4202 <E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the
 4203 uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the
 4204 portable character set are automatically included in this class.

4205 **blank** Define characters to be classified as <blank> characters.
 4206 In the POSIX locale, only the <space> and <tab> shall be included.

4207 In a locale definition file, the <space> and <tab> are automatically included in
 4208 this class.

4209 **charclass** Define one or more locale-specific character class names as strings separated
 4210 by <semicolon> characters. Each named character class can then be defined
 4211 subsequently in the *LC_CTYPE* definition. A character class name shall consist
 4212 of at least one and at most {CHARCLASS_NAME_MAX} bytes of
 4213 alphanumeric characters from the portable filename character set. The first
 4214 character of a character class name shall not be a digit. The name shall not
 4215 match any of the *LC_CTYPE* keywords defined in this volume of
 4216 POSIX.1-2017. Future versions of this standard will not specify any *LC_CTYPE*
 4217 keywords containing uppercase letters.

4218 *charclass-name* Define characters to be classified as belonging to the named locale-specific
 4219 character class. In the POSIX locale, locale-specific named character classes
 4220 need not exist.

4221 If a class name is defined by a **charclass** keyword, but no characters are
 4222 subsequently assigned to it, this is not an error; it represents a class without
 4223 any characters belonging to it.

4224 The *charclass-name* can be used as the *property* argument to the *wctype()*
 4225 function, in regular expression and shell pattern-matching bracket
 4226 expressions, and by the *tr* command.

4227 **toupper** Define the mapping of lowercase letters to uppercase letters.
 4228 In the POSIX locale, the 26 lowercase characters:
 4229 a b c d e f g h i j k l m n o p q r s t u v w x y z
 4230 shall be mapped to the corresponding 26 uppercase characters:
 4231 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 4232 In a locale definition file, the operand shall consist of character pairs,
 4233 separated by <semicolon> characters. The characters in each character pair
 4234 shall be separated by a <comma> and the pair enclosed by parentheses. The
 4235 first character in each pair is the lowercase letter, the second the corresponding
 4236 uppercase letter. Only characters specified for the keywords **lower** and **upper**
 4237 shall be specified. The lowercase letters <a> to <z>, and their corresponding
 4238 uppercase letters <A> to <Z>, of the portable character set are automatically
 4239 included in this mapping, but only when the **toupper** keyword is omitted
 4240 from the locale definition.

4241 **tolower** Define the mapping of uppercase letters to lowercase letters.
 4242 In the POSIX locale, the 26 uppercase characters:
 4243 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 4244 shall be mapped to the corresponding 26 lowercase characters:
 4245 a b c d e f g h i j k l m n o p q r s t u v w x y z
 4246 In a locale definition file, the operand shall consist of character pairs,
 4247 separated by <semicolon> characters. The characters in each character pair
 4248 shall be separated by a <comma> and the pair enclosed by parentheses. The
 4249 first character in each pair is the uppercase letter, the second the
 4250 corresponding lowercase letter. Only characters specified for the keywords
 4251 **lower** and **upper** shall be specified. If the **tolower** keyword is omitted from
 4252 the locale definition, the mapping is the reverse mapping of the one specified
 4253 for **toupper**.

4254 The following table shows the character class combinations allowed:

4255 **Table 7-1** Valid Character Class Combinations

In Class	Can Also Belong To												
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank		
upper			†	'	Ax	xx	x	A	A		†	'	x
lower			†	x	x	x	A	A	A		†	'	x
alpha		†	'		† x	' x	x x	A	A		†	'	x
digit	x	x	x		x	x	x	A	A	A	x		
space	x	x	x	x			†	*	*	x	†	'	
cntrl	x	x	x	x			†	x	' x	x x			†
punct	x	x	x	x		†	'	x	A	A	x		†
graph		†	'	†	†	'	†	†	'	'	x		†
print		†	'	†	†	'	†	†	'	'	x		†
xdigit		†	'	†	†	'	x	†	x'	xA	A		x
blank	x	x	x	x	A		†	'	* *	* *	*		x

4269 **Notes:**

- 4270 1. Explanation of codes:
- 4271 A Automatically included; see text.
- 4272 ‡ permitted.
- 4273 x Mutually-exclusive.
- 4274 * See note 2.
- 4275 2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or
- 4276 **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters
- 4277 can be classified as any of **punct**, **graph**, or **print**.

4278 7.3.1.1 *LC_CTYPE Category in the POSIX Locale*

4279 The minimum character classifications for the POSIX locale follow; the code listing depicts the

4280 *localedef* input, and the table represents the same information, sorted by character.

4281 Implementations may add additional characters to the **cntrl** and **punct** classifications but shall

4282 not make any other additions.

```

4283 LC_CTYPE
4284 # The following is the minimum POSIX locale LC_CTYPE.
4285 # "alpha" is by definition "upper" and "lower"
4286 # "alnum" is by definition "alpha" and "digit"
4287 # "print" is by definition "alnum", "punct", and the <space>
4288 # "graph" is by definition "alnum" and "punct"
4289 #
4290 upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4291          <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4292 #
4293 lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4294          <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4295 #
4296 digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4297          <seven>;<eight>;<nine>
4298 #
4299 space    <tab>;<newline>;<vertical-tab>;<form-feed>;\
4300          <carriage-return>;<space>
4301 #
4302 cntrl    <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4303          <form-feed>;<carriage-return>;\
4304          <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4305          <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4306          <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4307          <IS1>;<DEL>
4308 #
4309 punct    <exclamation-mark>;<quotation-mark>;<number-sign>;\
4310          <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4311          <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4312          <plus-sign>;<comma>;<hyphen-minus>;<period>;<slash>;\
4313          <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4314          <greater-than-sign>;<question-mark>;<commercial-at>;\
4315          <left-square-bracket>;<backslash>;<right-square-bracket>;\

```

```

4316         <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4317         <vertical-line>;<right-curly-bracket>;<tilde>
4318     #
4319     xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4320         <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4321     #
4322     blank   <space>;<tab>
4323     #
4324     toupper (<a>, <A>); (<b>, <B>); (<c>, <C>); (<d>, <D>); (<e>, <E>); \
4325         (<f>, <F>); (<g>, <G>); (<h>, <H>); (<i>, <I>); (<j>, <J>); \
4326         (<k>, <K>); (<l>, <L>); (<m>, <M>); (<n>, <N>); (<o>, <O>); \
4327         (<p>, <P>); (<q>, <Q>); (<r>, <R>); (<s>, <S>); (<t>, <T>); \
4328         (<u>, <U>); (<v>, <V>); (<w>, <W>); (<x>, <X>); (<y>, <Y>); (<z>, <Z>)
4329     #
4330     tolower (<A>, <a>); (<B>, <b>); (<C>, <c>); (<D>, <d>); (<E>, <e>); \
4331         (<F>, <f>); (<G>, <g>); (<H>, <h>); (<I>, <i>); (<J>, <j>); \
4332         (<K>, <k>); (<L>, <l>); (<M>, <m>); (<N>, <n>); (<O>, <o>); \
4333         (<P>, <p>); (<Q>, <q>); (<R>, <r>); (<S>, <s>); (<T>, <t>); \
4334         (<U>, <u>); (<V>, <v>); (<W>, <w>); (<X>, <x>); (<Y>, <y>); (<Z>, <z>)
4335     END LC_CTYPE

```

4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366

Symbolic Name	Other Case	Character Classes
<NUL>		cntrl
<SOH>		cntrl
<STX>		cntrl
<ETX>		cntrl
<EOT>		cntrl
<ENQ>		cntrl
<ACK>		cntrl
<alert>		cntrl
<backspace>		cntrl
<tab>		cntrl, space, blank
<newline>		cntrl, space
<vertical-tab>		cntrl, space
<form-feed>		cntrl, space
<carriage-return>		cntrl, space
<SO>		cntrl
<SI>		cntrl
<DLE>		cntrl
<DC1>		cntrl
<DC2>		cntrl
<DC3>		cntrl
<DC4>		cntrl
<NAK>		cntrl
<SYN>		cntrl
<ETB>		cntrl
<CAN>		cntrl
		cntrl
<SUB>		cntrl
<ESC>		cntrl
<IS4>		cntrl
<IS3>		cntrl

	Symbolic Name	Other Case	Character Classes
4367	<IS2>		cntrl
4368	<IS1>		cntrl
4369	<space>		space, print, blank
4370	<exclamation-mark>		punct, print, graph
4371	<quotation-mark>		punct, print, graph
4372	<number-sign>		punct, print, graph
4373	<dollar-sign>		punct, print, graph
4374	<percent-sign>		punct, print, graph
4375	<ampersand>		punct, print, graph
4376	<apostrophe>		punct, print, graph
4377	<left-parenthesis>		punct, print, graph
4378	<right-parenthesis>		punct, print, graph
4379	<asterisk>		punct, print, graph
4380	<plus-sign>		punct, print, graph
4381	<comma>		punct, print, graph
4382	<hyphen-minus>		punct, print, graph
4383	<period>		punct, print, graph
4384	<slash>		punct, print, graph
4385	<zero>		digit, xdigit, print, graph
4386	<one>		digit, xdigit, print, graph
4387	<two>		digit, xdigit, print, graph
4388	<three>		digit, xdigit, print, graph
4389	<four>		digit, xdigit, print, graph
4390	<five>		digit, xdigit, print, graph
4391	<six>		digit, xdigit, print, graph
4392	<seven>		digit, xdigit, print, graph
4393	<eight>		digit, xdigit, print, graph
4394	<nine>		digit, xdigit, print, graph
4395	<colon>		punct, print, graph
4396	<semicolon>		punct, print, graph
4397	<less-than-sign>		punct, print, graph
4398	<equals-sign>		punct, print, graph
4399	<greater-than-sign>		punct, print, graph
4400	<question-mark>		punct, print, graph
4401	<commercial-at>		punct, print, graph
4402	<A>	<a>	upper, xdigit, alpha, print, graph
4403			upper, xdigit, alpha, print, graph
4404	<C>	<c>	upper, xdigit, alpha, print, graph
4405	<D>	<d>	upper, xdigit, alpha, print, graph
4406	<E>	<e>	upper, xdigit, alpha, print, graph
4407	<F>	<f>	upper, xdigit, alpha, print, graph
4408	<G>	<g>	upper, alpha, print, graph
4409	<H>	<h>	upper, alpha, print, graph
4410	<I>	<i>	upper, alpha, print, graph
4411	<J>	<j>	upper, alpha, print, graph
4412	<K>	<k>	upper, alpha, print, graph
4413	<L>	<l>	upper, alpha, print, graph
4414	<M>	<m>	upper, alpha, print, graph
4415	<N>	<n>	upper, alpha, print, graph
4416	<O>	<o>	upper, alpha, print, graph
4417	<P>	<p>	upper, alpha, print, graph

	Symbolic Name	Other Case	Character Classes
4419			
4420	<Q>	<q>	upper, alpha, print, graph
4421	<R>	<r>	upper, alpha, print, graph
4422	<S>	<s>	upper, alpha, print, graph
4423	<T>	<t>	upper, alpha, print, graph
4424	<U>	<u>	upper, alpha, print, graph
4425	<V>	<v>	upper, alpha, print, graph
4426	<W>	<w>	upper, alpha, print, graph
4427	<X>	<x>	upper, alpha, print, graph
4428	<Y>	<y>	upper, alpha, print, graph
4429	<Z>	<z>	upper, alpha, print, graph
4430	<left-square-bracket>		punct, print, graph
4431	<backslash>		punct, print, graph
4432	<right-square-bracket>		punct, print, graph
4433	<circumflex>		punct, print, graph
4434	<underscore>		punct, print, graph
4435	<grave-accent>		punct, print, graph
4436	<a>	<A>	lower, xdigit, alpha, print, graph
4437			lower, xdigit, alpha, print, graph
4438	<c>	<C>	lower, xdigit, alpha, print, graph
4439	<d>	<D>	lower, xdigit, alpha, print, graph
4440	<e>	<E>	lower, xdigit, alpha, print, graph
4441	<f>	<F>	lower, xdigit, alpha, print, graph
4442	<g>	<G>	lower, alpha, print, graph
4443	<h>	<H>	lower, alpha, print, graph
4444	<i>	<I>	lower, alpha, print, graph
4445	<j>	<J>	lower, alpha, print, graph
4446	<k>	<K>	lower, alpha, print, graph
4447	<l>	<L>	lower, alpha, print, graph
4448	<m>	<M>	lower, alpha, print, graph
4449	<n>	<N>	lower, alpha, print, graph
4450	<o>	<O>	lower, alpha, print, graph
4451	<p>	<P>	lower, alpha, print, graph
4452	<q>	<Q>	lower, alpha, print, graph
4453	<r>	<R>	lower, alpha, print, graph
4454	<s>	<S>	lower, alpha, print, graph
4455	<t>	<T>	lower, alpha, print, graph
4456	<u>	<U>	lower, alpha, print, graph
4457	<v>	<V>	lower, alpha, print, graph
4458	<w>	<W>	lower, alpha, print, graph
4459	<x>	<X>	lower, alpha, print, graph
4460	<y>	<Y>	lower, alpha, print, graph
4461	<z>	<Z>	lower, alpha, print, graph
4462	<left-curly-bracket>		punct, print, graph
4463	<vertical-line>		punct, print, graph
4464	<right-curly-bracket>		punct, print, graph
4465	<tilde>		punct, print, graph
4466			cntrl

4467 **7.3.2 LC_COLLATE**

4468 The *LC_COLLATE* category provides a collation sequence definition for numerous utilities in the
 4469 Shell and Utilities volume of POSIX.1-2017 (*ls*, *sort*, and so on), regular expression matching (see
 4470 [Chapter 9](#), on page 181), and the *strcoll()*, *strxfrm()*, *wscoll()*, and *wcsxfrm()* functions in the
 4471 System Interfaces volume of POSIX.1-2017.

4472 A collation sequence definition shall define the relative order between collating elements
 4473 (characters and multi-character collating elements) in the locale. This order is expressed in terms
 4474 of collation values; that is, by assigning each element one or more collation values (also known
 4475 as collation weights). This does not imply that implementations shall assign such values, but
 4476 that ordering of strings using the resultant collation definition in the locale behaves as if such
 4477 assignment is done and used in the collation process. At least the following capabilities are
 4478 provided:

- 4479 1. **Multi-character collating elements.** Specification of multi-character collating elements
 4480 (that is, sequences of two or more characters to be collated as an entity).
- 4481 2. **User-defined ordering of collating elements.** Each collating element shall be assigned a
 4482 collation value defining its order in the character (or basic) collation sequence. This
 4483 ordering is used by regular expressions and pattern matching and, unless collation
 4484 weights are explicitly specified, also as the collation weight to be used in sorting.
- 4485 3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or
 4486 more (up to the limit {COLL_WEIGHTS_MAX}, as defined in [<limits.h>](#)) collating
 4487 weights for use in sorting. The first weight is hereafter referred to as the primary weight.
- 4488 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4489 5. **Equivalence class definition.** Two or more collating elements have the same collation
 4490 value (primary weight).
- 4491 6. **Ordering by weights.** When two strings are compared to determine their relative order,
 4492 the two strings are first broken up into a series of collating elements; the elements in each
 4493 successive pair of elements are then compared according to the relative primary weights
 4494 for the elements. If equal, and more than one weight has been assigned, then the pairs of
 4495 collating elements are re-compared according to the relative subsequent weights, until
 4496 either a pair of collating elements compare unequal or the weights are exhausted.

4497 All implementation-provided locales (either preinstalled or provided as locale definitions which
 4498 can be installed later) should define a collation sequence that has a total ordering of all
 4499 characters unless the locale name has an '@' modifier indicating that it has a special collation
 4500 sequence (for example, @icase could indicate that each upper and lowercase character pair
 4501 collates equally).

4502 **Notes:**

- 4503 1. A future version of this standard may require these locales to define a collation sequence
 4504 that has a total ordering of all characters (by changing ``should'' to ``shall'').
- 4505 2. Users installing their own locales should ensure that they define a collation sequence
 4506 with a total ordering of all characters unless an '@' modifier in the locale name (such as
 4507 @icase) indicates that it has a special collation sequence.

4508 The following keywords shall be recognized in a collation sequence definition. They are
 4509 described in detail in the following sections.

4510 **copy** Specify the name of an existing locale which shall be used as the
 4511 definition of this category. If this keyword is specified, no other keyword
 4512 shall be specified.

4513	collating-element	Define a collating-element symbol representing a multi-character collating element. This keyword is optional.
4514		
4515	collating-symbol	Define a collating symbol for use in collation order statements. This keyword is optional.
4516		
4517	order_start	Define collation rules. This statement shall be followed by one or more collation order statements, assigning character collation values and collation weights to collating elements.
4518		
4519		
4520	order_end	Specify the end of the collation-order statements.

4521 7.3.2.1 *The collating-element Keyword*

4522 In addition to the collating elements in the character set, the **collating-element** keyword can be
4523 used to define multi-character collating elements. The syntax is as follows:

```
4524 "collating-element %s from \"%s\"\\n", <collating-symbol>, <string>
```

4525 The *<collating-symbol>* operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A *<collating-element>* defined via this keyword is only recognized with the *LC_COLLATE* category.

4530 For example:

```
4531 collating-element <ch> from "<c><h>"
4532 collating-element <e-acute> from "<acute><e>"
4533 collating-element <ll> from "ll"
```

4534 7.3.2.2 *The collating-symbol Keyword*

4535 This keyword shall be used to define symbols for use in collation sequence statements; that is,
4536 between the **order_start** and the **order_end** keywords. The syntax is as follows:

```
4537 "collating-symbol %s\\n", <collating-symbol>
```

4538 The *<collating-symbol>* shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. A *<collating-symbol>* defined via this keyword is only recognized within the *LC_COLLATE* category.

4542 For example:

```
4543 collating-symbol <UPPER_CASE>
4544 collating-symbol <HIGH>
```

4545 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
4546 position in the character order sequence. While such a symbolic name does not represent any
4547 collating element, it can be used as a weight.

4548 7.3.2.3 *The order_start Keyword*

4549 The **order_start** keyword shall precede collation order entries and also define the number of
4550 weights for this collation sequence definition and other collation rules. The syntax is as follows:

4551 `"order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...`

4552 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
4553 applied when strings are compared. The number of operands define how many weights each
4554 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
4555 first operand defines rules to be applied when comparing strings using the first (primary)
4556 weight; the second when comparing strings using the second weight, and so on. Operands shall
4557 be separated by <semicolon> characters (';'). Each operand shall consist of one or more
4558 collation directives, separated by <comma> characters (','). If the number of operands exceeds
4559 the {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
4560 directives shall be supported:

4561 **forward** Specifies that comparison operations for the weight level shall proceed from start
4562 of string towards the end of string.

4563 **backward** Specifies that comparison operations for the weight level shall proceed from end of
4564 string towards the beginning of string.

4565 **position** Specifies that comparison operations for the weight level shall consider the relative
4566 position of elements in the strings not subject to **IGNORE**. The string containing
4567 an element not subject to **IGNORE** after the fewest collating elements subject to
4568 **IGNORE** from the start of the compare shall collate first. If both strings contain a
4569 character not subject to **IGNORE** in the same relative position, the collating values
4570 assigned to the elements shall determine the ordering. In case of equality,
4571 subsequent characters not subject to **IGNORE** shall be considered in the same
4572 manner.

4573 The directives **forward** and **backward** are mutually-exclusive.

4574 If no operands are specified, a single **forward** operand shall be assumed.

4575 For example:

4576 `order_start forward;backward`

4577 7.3.2.4 *Collation Order*

4578 The **order_start** keyword shall be followed by collating identifier entries. The syntax for the
4579 collating element entries is as follows:

4580 `"%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...`

4581 Each *collating-identifier* shall consist of either a character (in any of the forms defined in [Section](#)
4582 [7.3](#), on page 136), a *collating-element*, a *collating-symbol*, an ellipsis, or the special symbol
4583 **UNDEFINED**. The order in which collating elements are specified determines the character
4584 order sequence, such that each collating element shall compare less than the elements following
4585 it.

4586 A *collating-element* shall be used to specify multi-character collating elements, and indicates
4587 that the character sequence specified via the *collating-element* is to be collated as a unit and in
4588 the relative order specified by its place.

4589 A *collating-symbol* can be used to define a position in the relative order for use in weights. No
4590 weights shall be specified with a *collating-symbol*.

4591 The ellipsis symbol specifies that a sequence of characters shall collate according to their
 4592 encoded character values. It shall be interpreted as indicating that all characters with a coded
 4593 character set value higher than the value of the character in the preceding line, and lower than
 4594 the coded character set value for the character in the following line, in the current coded
 4595 character set, shall be placed in the character collation order between the previous and the
 4596 following character in ascending order according to their coded character set values. An initial
 4597 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing
 4598 ellipsis as if the following line specified the highest coded character set value in the current
 4599 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do
 4600 not specify characters in the current coded character set. The use of the ellipsis symbol ties the
 4601 definition to a specific coded character set and may preclude the definition from being portable
 4602 between implementations.

4603 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not
 4604 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character
 4605 collation order at the point indicated by the symbol, and in ascending order according to their
 4606 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded
 4607 character set contains characters not specified in this section, the utility shall issue a warning
 4608 message and place such characters at the end of the character collation order.

4609 The optional operands for each collation-element shall be used to define the primary, secondary,
 4610 or subsequent weights for the collating element. The first operand specifies the relative primary
 4611 weight, the second the relative secondary weight, and so on. Two or more collation-elements can
 4612 be assigned the same weight; they belong to the same "equivalence class" if they have the same
 4613 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**
 4614 are removed, unless the **position** collation directive is specified for the corresponding level with
 4615 the **order_start** keyword. Then each successive pair of elements shall be compared according to
 4616 the relative weights for the elements. If the two strings compare equal, the process shall be
 4617 repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

4618 Weights should be assigned such that the collation sequence has a total ordering of all characters
 4619 unless an '@' modifier in the locale name indicates that it has a special collation sequence.

4620 **Note:** A future version of this standard may require a total ordering of all characters for
 4621 implementation-provided locales that do not have an '@' modifier in the locale name. See
 4622 [Section 7.3.2](#) (on page 147).

4623 Weights shall be expressed as characters (in any of the forms specified in [Section 7.3](#), on page
 4624 136), *<collating-symbol>*s, *<collating-element>*s, an ellipsis, or the special symbol **IGNORE**. A
 4625 single character, a *<collating-symbol>*, or a *<collating-element>* shall represent the relative position
 4626 in the character collating sequence of the character or symbol, rather than the character or
 4627 characters themselves. Thus, rather than assigning absolute values to weights, a particular
 4628 weight is expressed using the relative order value assigned to a collating element based on its
 4629 order in the character collation sequence.

4630 One-to-many mapping is indicated by specifying two or more concatenated characters or
 4631 symbolic names. For example, if the *<eszet>* is given the string "*<s><s>*" as a weight,
 4632 comparisons are performed as if all occurrences of the *<eszet>* are replaced by "*<s><s>*"
 4633 (assuming that "*<s>*" has the collating weight "*<s>*"). If it is necessary to define *<eszet>* and
 4634 "*<s><s>*" as an equivalence class, then a collating element must be defined for the string "*ss*".

4635 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the
 4636 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special
 4637 symbol shall by default be assigned the same primary weight (that is, they belong to the same
 4638 equivalence class) if the collation order has more than one weight level. If the collation order has
 4639 only one weight level, these characters should be assigned unique primary weights, equal to the
 4640 relative order of their character in the character collation sequence, but may be assigned the

4641 same primary weight.

4642 **Note:** A future version of this standard may require these characters to be assigned unique primary
4643 weights if the collation order has only one weight level.

4644 An ellipsis symbol as a weight shall be interpreted to mean that each character in the sequence
4645 shall have unique weights, equal to the relative order of their character in the character collation
4646 sequence. The use of the ellipsis as a weight shall be treated as an error if the collating element is
4647 neither an ellipsis nor the special symbol **UNDEFINED**.

4648 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using
4649 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that
4650 is, as if the string did not contain the collating element. In regular expressions and pattern
4651 matching, all characters that are subject to **IGNORE** in their primary weight form an
4652 equivalence class.

4653 An empty operand shall be interpreted as the collating element itself.

4654 For example, the order statement:

```
4655 <a> <a>;<a>
```

4656 is equal to:

```
4657 <a>
```

4658 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be
4659 interpreted as the value of each character defined by the ellipsis.

4660 The collation order as defined in this section affects the interpretation of bracket expressions in
4661 regular expressions (see [Section 9.3.5](#), on page 184).

4662 For example:

```
4663 order_start forward;backward
4664 <LOW>
4665 <space> <LOW>;<space>
4666 ... <LOW>;...
4667 <a> <a>;<a>
4668 <a-acute> <a>;<a-acute>
4669 <a-grave> <a>;<a-grave>
4670 <A> <a>;<A>
4671 <A-acute> <a>;<A-acute>
4672 <A-grave> <a>;<A-grave>
4673 <ch> <ch>;<ch>
4674 <Ch> <ch>;<Ch>
4675 <s> <s>;<s>
4676 <eszet> "<s><s>";"<eszet><eszet>"
4677 UNDEFINED IGNORE;...
4678 order_end
```

4679 This example is interpreted as follows:

- 4680 1. All characters between <space> and 'a' shall have the same primary equivalence class
4681 and individual secondary weights based on their ordinal encoded values.
- 4682 2. All characters based on the uppercase or lowercase character 'a' belong to the same
4683 primary equivalence class.

- 4684 3. The multi-character collating element <ch> is represented by the collating symbol <ch>
 4685 and belongs to the same primary equivalence class as the multi-character collating
 4686 element <Ch>.
- 4687 4. The **UNDEFINED** means that all characters not specified in this definition (explicitly or
 4688 via the ellipsis) shall be ignored when comparing primary weights, and have individual
 4689 secondary weights based on their ordinal encoded values.

4690 7.3.2.5 *The order_end Keyword*

4691 The collating order entries shall be terminated with an **order_end** keyword.

4692 7.3.2.6 *LC_COLLATE Category in the POSIX Locale*

4693 The minimum collation sequence definition of the POSIX locale follows; the code listing depicts
 4694 the *localedef* input. All characters not explicitly listed here shall be inserted in the character
 4695 collation order after the listed characters and shall be assigned unique primary weights. If the
 4696 listed characters have ASCII encoding, the other characters shall be in ascending order according
 4697 to their coded character set values; otherwise, the order of the other characters is unspecified.
 4698 The collation sequence shall not include any multi-character collating elements.

```

4699           LC_COLLATE
4700           # This is the minimum input for the POSIX locale definition for the
4701           # LC_COLLATE category. Characters in this list are in the same order
4702           # as in the ASCII codeset.
4703           order_start forward
4704           <NUL>
4705           <SOH>
4706           <STX>
4707           <ETX>
4708           <EOT>
4709           <ENQ>
4710           <ACK>
4711           <alert>
4712           <backspace>
4713           <tab>
4714           <newline>
4715           <vertical-tab>
4716           <form-feed>
4717           <carriage-return>
4718           <SO>
4719           <SI>
4720           <DLE>
4721           <DC1>
4722           <DC2>
4723           <DC3>
4724           <DC4>
4725           <NAK>
4726           <SYN>
4727           <ETB>
4728           <CAN>
4729           <EM>
4730           <SUB>

```

4731 <ESC>
4732 <IS4>
4733 <IS3>
4734 <IS2>
4735 <IS1>
4736 <space>
4737 <exclamation-mark>
4738 <quotation-mark>
4739 <number-sign>
4740 <dollar-sign>
4741 <percent-sign>
4742 <ampersand>
4743 <apostrophe>
4744 <left-parenthesis>
4745 <right-parenthesis>
4746 <asterisk>
4747 <plus-sign>
4748 <comma>
4749 <hyphen-minus>
4750 <period>
4751 <slash>
4752 <zero>
4753 <one>
4754 <two>
4755 <three>
4756 <four>
4757 <five>
4758 <six>
4759 <seven>
4760 <eight>
4761 <nine>
4762 <colon>
4763 <semicolon>
4764 <less-than-sign>
4765 <equals-sign>
4766 <greater-than-sign>
4767 <question-mark>
4768 <commercial-at>
4769 <A>
4770
4771 <C>
4772 <D>
4773 <E>
4774 <F>
4775 <G>
4776 <H>
4777 <I>
4778 <J>
4779 <K>
4780 <L>
4781 <M>
4782 <N>
4783 <O>


```
4784      <P>
4785      <Q>
4786      <R>
4787      <S>
4788      <T>
4789      <U>
4790      <V>
4791      <W>
4792      <X>
4793      <Y>
4794      <Z>
4795      <left-square-bracket>
4796      <backslash>
4797      <right-square-bracket>
4798      <circumflex>
4799      <underscore>
4800      <grave-accent>
4801      <a>
4802      <b>
4803      <c>
4804      <d>
4805      <e>
4806      <f>
4807      <g>
4808      <h>
4809      <i>
4810      <j>
4811      <k>
4812      <l>
4813      <m>
4814      <n>
4815      <o>
4816      <p>
4817      <q>
4818      <r>
4819      <s>
4820      <t>
4821      <u>
4822      <v>
4823      <w>
4824      <x>
4825      <y>
4826      <z>
4827      <left-curly-bracket>
4828      <vertical-line>
4829      <right-curly-bracket>
4830      <tilde>
4831      <DEL>
4832      order_end
4833      #
4834      END LC_COLLATE
```

4835 **7.3.3 LC_MONETARY**

4836 The *LC_MONETARY* category shall define the rules and symbols that are used to format
4837 monetary numeric information.

4838 This information is available through the *localeconv()* function and is used by the *strfmon()*
4839 function.

4840 Some of the information is also available in an alternative form via the *nl_langinfo()* function
4841 (see CRNCYSTR in [<langinfo.h>](#)).

4842 The following items are defined in this category of the locale. The item names are the keywords
4843 recognized by the *localedef* utility when defining a locale. They are also similar to the member
4844 names of the *lconv* structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the
4845 header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the
4846 empty string (" ") for unspecified or size zero string items.

4847 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
4848 [Section 7.4](#) (on page 166). For some keywords, the strings can contain only integers. Keywords
4849 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
4850 are used to indicate that the value is not available in the locale. The following keywords shall be
4851 recognized:

4852 **copy** Specify the name of an existing locale which shall be used as the
4853 definition of this category. If this keyword is specified, no other keyword
4854 shall be specified.

4855 **Note:** This is a *localedef* utility keyword, unavailable through *localeconv()*.

4856 **int_curr_symbol** The international currency symbol. The operand shall be a four-character
4857 string, with the first three characters containing the alphabetic
4858 international currency symbol. The international currency symbol should
4859 be chosen in accordance with those specified in the ISO 4217 standard.
4860 The fourth character shall be the character used to separate the
4861 international currency symbol from the monetary quantity.

4862 **currency_symbol** The string that shall be used as the local currency symbol.

4863 **mon_decimal_point** The operand is a string containing the symbol that shall be used as the
4864 decimal delimiter (radix character) in monetary formatted quantities.

4865 **mon_thousands_sep** The operand is a string containing the symbol that shall be used as a
4866 separator for groups of digits to the left of the decimal delimiter in
4867 formatted monetary quantities.

4868 **mon_grouping** Define the size of each group of digits in formatted monetary quantities.
4869 The operand is a sequence of integers separated by [<semicolon>](#)
4870 characters. Each integer specifies the number of digits in each group, with
4871 the initial integer defining the size of the group immediately preceding
4872 the decimal delimiter, and the following integers defining the preceding
4873 groups. If the last integer is not -1, then the size of the previous group (if
4874 any) shall be repeatedly used for the remainder of the digits. If the last
4875 integer is -1, then no further grouping shall be performed.

4876 **positive_sign** A string that shall be used to indicate a non-negative-valued formatted
4877 monetary quantity.

4878 **negative_sign** A string that shall be used to indicate a negative-valued formatted
4879 monetary quantity.

4880	int_frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using int_curr_symbol .
4881		
4882		
4883	frac_digits	An integer representing the number of fractional digits (those to the right of the decimal delimiter) to be written in a formatted monetary quantity using currency_symbol .
4884		
4885		
4886	p_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4887		
4888		
4889	p_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a non-negative formatted monetary quantity.
4890		
4891		The values of p_sep_by_space , n_sep_by_space , int_p_sep_by_space , and int_n_sep_by_space are interpreted according to the following:
4892		
4893		0 No <space> separates the currency symbol and value.
4894		1 If the currency symbol and sign string are adjacent, a <space> separates them from the value; otherwise, a <space> separates the currency symbol from the value.
4895		
4896		
4897		2 If the currency symbol and sign string are adjacent, a <space> separates them; otherwise, a <space> separates the sign string from the value.
4898		
4899		
4900	n_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4901		
4902		
4903	n_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a negative formatted monetary quantity.
4904		
4905	p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a monetary quantity with a non-negative value. The following integer values shall be recognized for int_n_sign_posn , int_p_sign_posn , n_sign_posn , and p_sign_posn :
4906		
4907		
4908		
4909		0 Parentheses enclose the quantity and the currency_symbol .
4910		1 The sign string precedes the quantity and the currency_symbol .
4911		2 The sign string succeeds the quantity and the currency_symbol .
4912		3 The sign string precedes the currency_symbol .
4913		4 The sign string succeeds the currency_symbol .
4914	n_sign_posn	An integer set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity.
4915		
4916	int_p_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4917		
4918		
4919	int_n_cs_precedes	An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4920		
4921		

4922	int_p_sep_by_space	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the value for a non-negative internationally formatted monetary quantity.
4923		
4924		
4925	int_n_sep_by_space	Set to a value indicating the separation of the int_curr_symbol , the sign string, and the value for a negative internationally formatted monetary quantity.
4926		
4927		
4928	int_p_sign_posn	An integer set to a value indicating the positioning of the positive_sign for a positive monetary quantity formatted with the international format.
4929		
4930	int_n_sign_posn	An integer set to a value indicating the positioning of the negative_sign for a negative monetary quantity formatted with the international format.
4931		

4932 7.3.3.1 LC_MONETARY Category in the POSIX Locale

4933 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the
 4934 *localedef* input, the table representing the same information with the addition of *localeconv()* and
 4935 *nl_langinfo()* formats. All values are unspecified in the POSIX locale.

```

4936 LC_MONETARY
4937 # This is the POSIX locale definition for
4938 # the LC_MONETARY category.
4939 #
4940 int_curr_symbol      ""
4941 currency_symbol     ""
4942 mon_decimal_point   ""
4943 mon_thousands_sep  ""
4944 mon_grouping        -1
4945 positive_sign       ""
4946 negative_sign       ""
4947 int_frac_digits     -1
4948 frac_digits         -1
4949 p_cs_precedes       -1
4950 p_sep_by_space      -1
4951 n_cs_precedes       -1
4952 n_sep_by_space      -1
4953 p_sign_posn         -1
4954 n_sign_posn         -1
4955 int_p_cs_precedes   -1
4956 int_p_sep_by_space  -1
4957 int_n_cs_precedes   -1
4958 int_n_sep_by_space  -1
4959 int_p_sign_posn     -1
4960 int_n_sign_posn     -1
4961 #
4962 END LC_MONETARY

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
int_curr_symbol		‡ /A	' "" N	""
currency_symbol	CRNCYSTR	N/A	""	""
mon_decimal_point		‡ /A	' "" N	""
mon_thousands_sep		‡ /A	' "" N	""
mon_grouping		‡ /A	' "" N	-1
positive_sign		‡ /A	' "" N	""
negative_sign		‡ /A	' "" N	""
int_frac_digits		‡ /A	{CHAR_MAX}N	-1
frac_digits		‡ /A	{CHAR_MAX}N	-1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1
p_sep_by_space		‡ /A	{CHAR_MAX}N	-1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1
n_sep_by_space		‡ /A	{CHAR_MAX}N	-1
p_sign_posn		‡ /A	{CHAR_MAX}N	-1
n_sign_posn		‡ /A	{CHAR_MAX}N	-1
int_p_cs_precedes		‡ /A	{CHAR_MAX}N	-1
int_p_sep_by_space		‡ /A	{CHAR_MAX}N	-1
int_n_cs_precedes		‡ /A	{CHAR_MAX}N	-1
int_n_sep_by_space		‡ /A	{CHAR_MAX}N	-1
int_p_sign_posn		‡ /A	{CHAR_MAX}N	-1
int_n_sign_posn		‡ /A	{CHAR_MAX}N	-1

The entry N/A indicates that the value is not available in the POSIX locale.

7.3.4 LC_NUMERIC

The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-monetary numeric information. This information is available through the *localeconv()* function.

Some of the information is also available in an alternative form via the *nl_langinfo()* function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in *<locale.h>*; see *<locale.h>* for the exact symbols in the header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the empty string (" ") for unspecified or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in Section 7.4 (on page 166). For some keywords, the strings can only contain integers. Keywords that are not provided, string values set to the empty string (" "), or integer keywords set to -1, shall be used to indicate that the value is not available in the locale. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv()*.

decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the **decimal_point** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

5009 **thousands_sep** The operand is a string containing the symbol that shall be used as a separator
 5010 for groups of digits to the left of the decimal delimiter in numeric, non-
 5011 monetary formatted monetary quantities. In contexts where standards limit
 5012 the **thousands_sep** to a single byte, the result of specifying a multi-byte
 5013 operand shall be unspecified.

5014 **grouping** Define the size of each group of digits in formatted non-monetary quantities.
 5015 The operand is a sequence of integers separated by <semicolon> characters.
 5016 Each integer specifies the number of digits in each group, with the initial
 5017 integer defining the size of the group immediately preceding the decimal
 5018 delimiter, and the following integers defining the preceding groups. If the last
 5019 integer is not -1, then the size of the previous group (if any) shall be
 5020 repeatedly used for the remainder of the digits. If the last integer is -1, then no
 5021 further grouping shall be performed.

5022 7.3.4.1 *LC_NUMERIC Category in the POSIX Locale*

5023 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing
 5024 depicting the *localedef* input, the table representing the same information with the addition of
 5025 *localeconv()* values, and *nl_langinfo()* constants.

```
5026   LC_NUMERIC
5027   # This is the POSIX locale definition for
5028   # the LC_NUMERIC category.
5029   #
5030   decimal_point   "<period>"
5031   thousands_sep   ""
5032   grouping       -1
5033   #
5034   END LC_NUMERIC
```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	."	."	.
thousands_sep	THOUSEP	N/A	""	""
grouping		‡ /A	' ""	N -1

5040 The entry N/A indicates that the value is not available in the POSIX locale.

5041 **7.3.5 LC_TIME**

5042 The *LC_TIME* category shall define the interpretation of the conversion specifications supported
 5043 by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and
 5044 *nl_langinfo()* functions. Since the interfaces for C-language access and locale definition differ
 5045 significantly, they are described separately.

5046 7.3.5.1 *LC_TIME Locale Definition*

5047 In a locale definition, the following mandatory keywords shall be recognized:

5048 **copy** Specify the name of an existing locale which shall be used as the definition of
 5049 this category. If this keyword is specified, no other keyword shall be specified.

5050	abday	Define the abbreviated weekday names, corresponding to the %a conversion specification (conversion specification in the <i>strftime()</i> , <i>wcsftime()</i> , and <i>strptime()</i> functions). The operand shall consist of seven <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the day corresponding to Sunday, the second the abbreviated name of the day corresponding to Monday, and so on.
5051		
5052		
5053		
5054		
5055		
5056	day	Define the full weekday names, corresponding to the %A conversion specification. The operand shall consist of seven <semicolon>-separated strings, each surrounded by double-quotes. The first string is the full name of the day corresponding to Sunday, the second the full name of the day corresponding to Monday, and so on.
5057		
5058		
5059		
5060		
5061	abmon	Define the abbreviated month names, corresponding to the %b conversion specification. The operand shall consist of twelve <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the abbreviated name of the first month of the year (January), the second the abbreviated name of the second month, and so on.
5062		
5063		
5064		
5065		
5066	mon	Define the full month names, corresponding to the %B conversion specification. The operand shall consist of twelve <semicolon>-separated strings, each surrounded by double-quotes. The first string shall be the full name of the first month of the year (January), the second the full name of the second month, and so on.
5067		
5068		
5069		
5070		
5071	d_t_fmt	Define the appropriate date and time representation, corresponding to the %c conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in the table in Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').
5072		
5073		
5074		
5075		
5076	d_fmt	Define the appropriate date representation, corresponding to the %x conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 121).
5077		
5078		
5079		
5080	t_fmt	Define the appropriate time representation, corresponding to the %X conversion specification. The operand shall consist of a string containing any combination of characters and conversion specifications. In addition, the string can contain escape sequences defined in Table 5-1 (on page 121).
5081		
5082		
5083		
5084	am_pm	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i> strings, corresponding to the %p conversion specification. The operand shall consist of two strings, separated by a <semicolon>, each surrounded by double-quotes. The first string shall represent the <i>ante-meridiem</i> designation, the last string the <i>post-meridiem</i> designation.
5085		
5086		
5087		
5088		
5089	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with am_pm , corresponding to the %r conversion specification. The operand shall consist of a string and can contain any combination of characters and conversion specifications. If the string is empty, the 12-hour format is not supported in the locale.
5090		
5091		
5092		
5093		
5094	era	Define how years are counted and displayed for each era in a locale. The operand shall consist of <semicolon>-separated strings. Each string shall be an era description segment with the format:
5095		
5096		
5097		<i>direction:offset:start_date:end_date:era_name:era_format</i>

5098		according to the definitions below. There can be as many era description
5099		segments as are necessary to describe the different eras.
5100	Note:	The start of an era might not be the earliest point in the era. It may be the
5101		latest. For example, the Christian era BC starts on the day before January 1,
5102		AD 1, and increases with earlier time.
5103	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate
5104		that years closer to the <i>start_date</i> have lower numbers than those
5105		closer to the <i>end_date</i> . The '-' character shall indicate that years
5106		closer to the <i>start_date</i> have higher numbers than those closer to
5107		the <i>end_date</i> .
5108	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era,
5109		corresponding to the %EY conversion specification.
5110	<i>start_date</i>	A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the
5111		year, month, and day numbers respectively of the start of the era.
5112		Years prior to AD 1 shall be represented as negative numbers.
5113	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
5114		or one of the two special values "-*" or "+*". The value "-*"
5115		shall indicate that the ending date is the beginning of time. The
5116		value "+*" shall indicate that the ending date is the end of time.
5117	<i>era_name</i>	A string representing the name of the era, corresponding to the
5118		%EC conversion specification.
5119	<i>era_format</i>	A string for formatting the year in the era, corresponding to the
5120		%EY conversion specification.
5121	era_d_fmt	Define the format of the date in alternative era notation, corresponding to the
5122		%Ex conversion specification.
5123	era_t_fmt	Define the locale's appropriate alternative time format, corresponding to the
5124		%EX conversion specification.
5125	era_d_t_fmt	Define the locale's appropriate alternative date and time format,
5126		corresponding to the %Ec conversion specification.
5127	alt_digits	Define alternative symbols for digits, corresponding to the %O modified
5128		conversion specification. The operand shall consist of <semicolon>-separated
5129		strings, each surrounded by double-quotes. The first string shall be the
5130		alternative symbol corresponding with zero, the second string the symbol
5131		corresponding with one, and so on. Up to 100 alternative symbol strings can
5132		be specified. The %O modifier shall indicate that the string corresponding to
5133		the value specified via the conversion specification shall be used instead of the
5134		value.
5135	7.3.5.2	<i>LC_TIME C-Language Access</i>
5136		The following constants used to identify items of <i>langinfo</i> data can be used as arguments to the
5137		<i>nl_langinfo()</i> function to access information in the <i>LC_TIME</i> category. These constants are
5138		defined in the <langinfo.h> header.
5139	ABDAY_x	The abbreviated weekday names (for example, Sun), where <i>x</i> is a number
5140		from 1 to 7.

5141	DAY_x	The full weekday names (for example, Sunday), where <i>x</i> is a number from 1 to 7.
5142		
5143	ABMON_x	The abbreviated month names (for example, Jan), where <i>x</i> is a number from 1 to 12.
5144		
5145	MON_x	The full month names (for example, January), where <i>x</i> is a number from 1 to 12.
5146		
5147	D_T_FMT	The appropriate date and time representation.
5148	D_FMT	The appropriate date representation.
5149	T_FMT	The appropriate time representation.
5150	AM_STR	The appropriate ante-meridiem affix.
5151	PM_STR	The appropriate post-meridiem affix.
5152	T_FMT_AMP	The appropriate time representation in the 12-hour clock format with AM_STR and PM_STR.
5153		
5154	ERA	The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:
5155		
5156		
5157		<i>direction:offset:start_date:end_date:era_name:era_format</i>
5158		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras. Era description segments are separated by <semicolon> characters.
5159		
5160		
5161		<i>direction</i> Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
5162		
5163		
5164		
5165		
5166		<i>offset</i> The number of the year closest to the <i>start_date</i> in the era.
5167		<i>start_date</i> A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
5168		
5169		
5170		<i>end_date</i> The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
5171		
5172		
5173		
5174		<i>era_name</i> The era, corresponding to the %EC conversion specification.
5175		<i>era_format</i> The format of the year in the era, corresponding to the %EY conversion specification.
5176		
5177	ERA_D_FMT	The era date format.
5178	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the %EX conversion specification.
5179		
5180	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to the %EC conversion specification.
5181		

5182 ALT_DIGITS The alternative symbols for digits, corresponding to the %O conversion
 5183 specification modifier. The value consists of <semicolon>-separated symbols.
 5184 The first is the alternative symbol corresponding to zero, the second is the
 5185 symbol corresponding to one, and so on. Up to 100 alternative symbols may
 5186 be specified.

5187 7.3.5.3 LC_TIME Category in the POSIX Locale

5188 The LC_TIME category definition of the POSIX locale follows; the code listing depicts the
 5189 *localedef* input; the table represents the same information with the addition of *localedef* keywords,
 5190 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*
 5191 functions, and *nl_langinfo()* constants.

```

5192 LC_TIME
5193 # This is the POSIX locale definition for
5194 # the LC_TIME category.
5195 #
5196 # Abbreviated weekday names (%a)
5197 abday      "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
5198           "<T><h><u>";"<F><r><i>";"<S><a><t>"
5199 #
5200 # Full weekday names (%A)
5201 day        "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
5202           "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
5203           "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
5204           "<S><a><t><u><r><d><a><y>"
5205 #
5206 # Abbreviated month names (%b)
5207 abmon      "<J><a><n>";"<F><e><b>";"<M><a><r>";\
5208           "<A><p><r>";"<M><a><y>";"<J><u><n>";\
5209           "<J><u><l>";"<A><u><g>";"<S><e><p>";\
5210           "<O><c><t>";"<N><o><v>";"<D><e><c>"
5211 #
5212 # Full month names (%B)
5213 mon        "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
5214           "<M><a><r><c><h>";"<A><p><r><i><l>";\
5215           "<M><a><y>";"<J><u><n><e>";\
5216           "<J><u><l><y>";"<A><u><g><u><s><t>";\
5217           "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
5218           "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
5219 #
5220 # Equivalent of AM/PM (%p)      "AM";"PM"
5221 am_pm      "<A><M>";"<P><M>"
5222 #
5223 # Appropriate date and time representation (%c)
5224 #      "%a %b %e %H:%M:%S %Y"
5225 d_t_fmt    "<percent-sign><a><space><percent-sign><b>\
5226           <space><percent-sign><e><space><percent-sign><H>\
5227           <colon><percent-sign><M><colon><percent-sign><S>\
5228           <space><percent-sign><Y>"
5229 #
5230 # Appropriate date representation (%x)  "%m/%d/%y"
5231 d_fmt      "<percent-sign><m><slash><percent-sign><d>\

```

```

5232 <slash><percent-sign><y>"
5233 #
5234 # Appropriate time representation (%X) "%H:%M:%S"
5235 t_fmt "<percent-sign><H><colon><percent-sign><M>\
5236 <colon><percent-sign><S>"
5237 #
5238 # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
5239 t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
5240 <percent-sign><S><space><percent-sign><p>"
5241 #
5242 END LC_TIME

```

5243	localedef	langinfo	Conversion	POSIX
5244	Keyword	Constant	Specification	Locale Value
5245	d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
5246	d_fmt	D_FMT	%x	"%m/%d/%y"
5247	t_fmt	T_FMT	%X	"%H:%M:%S"
5248	am_pm	AM_STR	%p	"AM"
5249	am_pm	PM_STR	%p	"PM"
5250	t_fmt_ampm	T_FMT_AMPM	%r	"%I:%M:%S %p"
5251	day	DAY_1	%A	"Sunday"
5252	day	DAY_2	%A	"Monday"
5253	day	DAY_3	%A	"Tuesday"
5254	day	DAY_4	%A	"Wednesday"
5255	day	DAY_5	%A	"Thursday"
5256	day	DAY_6	%A	"Friday"
5257	day	DAY_7	%A	"Saturday"
5258	abday	ABDAY_1	%a	"Sun"
5259	abday	ABDAY_2	%a	"Mon"
5260	abday	ABDAY_3	%a	"Tue"
5261	abday	ABDAY_4	%a	"Wed"
5262	abday	ABDAY_5	%a	"Thu"
5263	abday	ABDAY_6	%a	"Fri"
5264	abday	ABDAY_7	%a	"Sat"
5265	mon	MON_1	%B	"January"
5266	mon	MON_2	%B	"February"
5267	mon	MON_3	%B	"March"
5268	mon	MON_4	%B	"April"
5269	mon	MON_5	%B	"May"
5270	mon	MON_6	%B	"June"
5271	mon	MON_7	%B	"July"
5272	mon	MON_8	%B	"August"
5273	mon	MON_9	%B	"September"
5274	mon	MON_10	%B	"October"
5275	mon	MON_11	%B	"November"
5276	mon	MON_12	%B	"December"
5277	abmon	ABMON_1	%b	"Jan"
5278	abmon	ABMON_2	%b	"Feb"
5279	abmon	ABMON_3	%b	"Mar"
5280	abmon	ABMON_4	%b	"Apr"
5281	abmon	ABMON_5	%b	"May"
5282	abmon	ABMON_6	%b	"Jun"

	localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
5283	abmon	ABMON_7	%b	"Jul "
5284	abmon	ABMON_8	%b	"Aug "
5285	abmon	ABMON_9	%b	"Sep "
5286	abmon	ABMON_10	%b	"Oct "
5287	abmon	ABMON_11	%b	"Nov "
5288	abmon	ABMON_12	%b	"Dec "
5289	era	ERA	%EC, %Ey, %EY	N/A
5290	era_d_fmt	ERA_D_FMT	%Ex	N/A
5291	era_t_fmt	ERA_T_FMT	%EX	N/A
5292	era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
5293	alt_digits	ALT_DIGITS	%O	N/A

5296 The entry N/A indicates the value is not available in the POSIX locale.

5297 **7.3.6 LC_MESSAGES**

5298 The *LC_MESSAGES* category shall define the format and values used by various utilities for
 5299 affirmative and negative responses. This information is available through the *nl_langinfo()*
 5300 function.

5301 The message catalog used by the standard utilities and selected by the *catopen()* function shall be
 5302 determined by the setting of *NLSPATH*; see [Chapter 8](#) (on page 173). The *LC_MESSAGES*
 5303 category can be specified as part of an *NLSPATH* substitution field.

5304 The following keywords shall be recognized as part of the locale definition file.

5305 **copy** Specify the name of an existing locale which shall be used as the definition of this
 5306 category. If this keyword is specified, no other keyword shall be specified.

5307 **Note:** This is a *localedef* keyword, unavailable through *nl_langinfo()*.

5308 **yesexpr** The operand consists of an extended regular expression (see [Section 9.4](#), on page
 5309 188) that describes acceptable affirmative responses to a question expecting an
 5310 affirmative or negative response.

5311 **noexpr** The operand consists of an extended regular expression that describes acceptable
 5312 negative responses to a question expecting an affirmative or negative response.

5313 **7.3.6.1 LC_MESSAGES Category in the POSIX Locale**

5314 The format and values for affirmative and negative responses of the POSIX locale follow; the
 5315 code listing depicting the *localedef* input, the table representing the same information with the
 5316 addition of *nl_langinfo()* constants.

```

5317 LC_MESSAGES
5318 # This is the POSIX locale definition for
5319 # the LC_MESSAGES category.
5320 #
5321 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
5322 #
5323 noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
5324 #
5325 END LC_MESSAGES
    
```

5326
5327
5328

localedef Keyword	langinfo Constant	POSIX Locale Value
yesexpr	YESEXPR	"^[yY]"
noexpr	NOEXPR	"^[nN]"

5329 **7.4 Locale Definition Grammar**

5330 The grammar and lexical conventions in this section shall together describe the syntax for the
5331 locale definition source. The general conventions for this style of grammar are described in XCU
5332 [Section 1.3](#) (on page 2335). The grammar shall take precedence over the text in this chapter.

5333 **7.4.1 Locale Lexical Conventions**

5334 The lexical conventions for the locale definition grammar are described in this section.

5335 The following tokens shall be processed (in addition to those string constants shown in the
5336 grammar):

- 5337 **LOC_NAME** A string of characters representing the name of a locale.
- 5338 **CHAR** Any single character.
- 5339 **NUMBER** A decimal number, represented by one or more decimal digits.
- 5340 **COLLSYMBOL** A symbolic name, enclosed between angle brackets. The string
5341 cannot duplicate any charmap symbol defined in the current
5342 charmap (if any), or a **COLLELEMENT** symbol.
- 5343 **COLLELEMENT** A symbolic name, enclosed between angle brackets, which cannot
5344 duplicate either any charmap symbol or a **COLLSYMBOL** symbol.
- 5345 **CHARCLASS** A string of alphanumeric characters from the portable character set,
5346 the first of which is not a digit, consisting of at least one and at most
5347 {CHARCLASS_NAME_MAX} bytes, and optionally surrounded by
5348 double-quotes.
- 5349 **CHARSYMBOL** A symbolic name, enclosed between angle brackets, from the current
5350 charmap (if any).
- 5351 **OCTAL_CHAR** One or more octal representations of the encoding of each byte in a
5352 single character. The octal representation consists of an escape
5353 character (normally a <backslash>) followed by two or more octal
5354 digits.
- 5355 **HEX_CHAR** One or more hexadecimal representations of the encoding of each
5356 byte in a single character. The hexadecimal representation consists of
5357 an escape character followed by the constant *x* and two or more
5358 hexadecimal digits.
- 5359 **DECIMAL_CHAR** One or more decimal representations of the encoding of each byte in
5360 a single character. The decimal representation consists of an escape
5361 character followed by a character 'd' and two or more decimal
5362 digits.

5363	ELLIPSIS	The string "...".
5364	EXTENDED_REG_EXP	An extended regular expression as defined in the grammar in Section 9.5 (on page 192).
5365		
5366	EOL	The line termination character <newline>.

5367 7.4.2 Locale Grammar

5368 This section presents the grammar for the locale definition.

```

5369 %token          LOC_NAME
5370 %token          CHAR
5371 %token          NUMBER
5372 %token          COLLSYMBOL COLLELEMENT
5373 %token          CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5374 %token          ELLIPSIS
5375 %token          EXTENDED_REG_EXP
5376 %token          EOL
5377 %start          locale_definition
5378 %%
5379 locale_definition : global_statements locale_categories
5380                  |                   locale_categories
5381                  ;
5382 global_statements : global_statements symbol_redefine
5383                  | symbol_redefine
5384                  ;
5385 symbol_redefine   : 'escape_char' CHAR EOL
5386                  | 'comment_char' CHAR EOL
5387                  ;
5388 locale_categories : locale_categories locale_category
5389                  | locale_category
5390                  ;
5391 locale_category  : lc_ctype | lc_collate | lc_messages
5392                  | lc_monetary | lc_numeric | lc_time
5393                  ;
5394 /* The following grammar rules are common to all categories */
5395 char_list        : char_list char_symbol
5396                  | char_symbol
5397                  ;
5398 char_symbol      : CHAR | CHARSYMBOL
5399                  | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5400                  ;
5401 elem_list        : elem_list char_symbol
5402                  | elem_list COLLSYMBOL
5403                  | elem_list COLLELEMENT
5404                  | char_symbol
5405                  | COLLSYMBOL

```

```

5406             | COLLELEMENT
5407             ;
5408     symb_list      : symb_list COLLSYMBOL
5409                   | COLLSYMBOL
5410                   ;
5411     locale_name    : LOC_NAME
5412                   | '"' LOC_NAME '"'
5413                   ;
5414     /* The following is the LC_CTYPE category grammar */
5415     lc_ctype       : ctype_hdr ctype_keywords      ctype_tlr
5416                   | ctype_hdr 'copy' locale_name EOL ctype_tlr
5417                   ;
5418     ctype_hdr      : 'LC_CTYPE' EOL
5419                   ;
5420     ctype_keywords : ctype_keywords ctype_keyword
5421                   | ctype_keyword
5422                   ;
5423     ctype_keyword  : charclass_keyword charclass_list EOL
5424                   | charconv_keyword charconv_list EOL
5425                   | 'charclass' charclass_namelist EOL
5426                   ;
5427     charclass_namelist : charclass_namelist ';' CHARCLASS
5428                   | CHARCLASS
5429                   ;
5430     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5431                   | 'punct' | 'xdigit' | 'space' | 'print'
5432                   | 'graph' | 'blank' | 'cntrl' | 'alnum'
5433                   | CHARCLASS
5434                   ;
5435     charclass_list  : charclass_list ';' char_symbol
5436                   | charclass_list ';' ELLIPSIS ';' char_symbol
5437                   | char_symbol
5438                   ;
5439     charconv_keyword : 'toupper'
5440                   | 'tolower'
5441                   ;
5442     charconv_list   : charconv_list ';' charconv_entry
5443                   | charconv_entry
5444                   ;
5445     charconv_entry  : '(' char_symbol ',' char_symbol ')'
5446                   ;
5447     ctype_tlr      : 'END' 'LC_CTYPE' EOL
5448                   ;
5449     /* The following is the LC_COLLATE category grammar */
5450     lc_collate     : collate_hdr collate_keywords      collate_tlr

```

```

5451         | collate_hdr 'copy' locale_name EOL collate_tlr
5452         ;
5453     collate_hdr      : 'LC_COLLATE' EOL
5454         ;
5455     collate_keywords :          order_statements
5456         | opt_statements order_statements
5457         ;
5458     opt_statements   : opt_statements collating_symbols
5459         | opt_statements collating_elements
5460         | collating_symbols
5461         | collating_elements
5462         ;
5463     collating_symbols : 'collating-symbol' COLLSYMBOL EOL
5464         ;
5465     collating_elements : 'collating-element' COLLELEMENT
5466         | 'from' "" elem_list "" EOL
5467         ;
5468     order_statements : order_start collation_order order_end
5469         ;
5470     order_start      : 'order_start' EOL
5471         | 'order_start' order_opts EOL
5472         ;
5473     order_opts       : order_opts ';' order_opt
5474         | order_opt
5475         ;
5476     order_opt        : order_opt ',' opt_word
5477         | opt_word
5478         ;
5479     opt_word         : 'forward' | 'backward' | 'position'
5480         ;
5481     collation_order   : collation_order collation_entry
5482         | collation_entry
5483         ;
5484     collation_entry   : COLLSYMBOL EOL
5485         | collation_element weight_list EOL
5486         | collation_element          EOL
5487         ;
5488     collation_element : char_symbol
5489         | COLLELEMENT
5490         | ELLIPSIS
5491         | 'UNDEFINED'
5492         ;
5493     weight_list       : weight_list ';' weight_symbol
5494         | weight_list ';'
5495         | weight_symbol
5496         ;

```



```

5497     weight_symbol      : /* empty */
5498     | char_symbol
5499     | COLLSYMBOL
5500     | '"' elem_list '"'
5501     | '"' symb_list '"'
5502     | ELLIPSIS
5503     | 'IGNORE'
5504     ;
5505     order_end           : 'order_end' EOL
5506     ;
5507     collate_tlr        : 'END' 'LC_COLLATE' EOL
5508     ;
5509     /* The following is the LC_MESSAGES category grammar */
5510     lc_messages         : messages_hdr messages_keywords      messages_tlr
5511     | messages_hdr 'copy' locale_name EOL messages_tlr
5512     ;
5513     messages_hdr       : 'LC_MESSAGES' EOL
5514     ;
5515     messages_keywords  : messages_keywords messages_keyword
5516     | messages_keyword
5517     ;
5518     messages_keyword   : 'yesexpr' '"' EXTENDED_REG_EXP '"' EOL
5519     | 'noexpr'  '"' EXTENDED_REG_EXP '"' EOL
5520     ;
5521     messages_tlr      : 'END' 'LC_MESSAGES' EOL
5522     ;
5523     /* The following is the LC_MONETARY category grammar */
5524     lc_monetary        : monetary_hdr monetary_keywords      monetary_tlr
5525     | monetary_hdr 'copy' locale_name EOL monetary_tlr
5526     ;
5527     monetary_hdr       : 'LC_MONETARY' EOL
5528     ;
5529     monetary_keywords  : monetary_keywords monetary_keyword
5530     | monetary_keyword
5531     ;
5532     monetary_keyword   : mon_keyword_string mon_string EOL
5533     | mon_keyword_char NUMBER EOL
5534     | mon_keyword_char '-1' EOL
5535     | mon_keyword_grouping mon_group_list EOL
5536     ;
5537     mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5538     | 'mon_decimal_point' | 'mon_thousands_sep'
5539     | 'positive_sign' | 'negative_sign'
5540     ;
5541     mon_string         : '"' char_list '"'

```

```

5542         | '""'
5543         ;
5544     mon_keyword_char : 'int_frac_digits' | 'frac_digits'
5545         | 'p_cs_precedes' | 'p_sep_by_space'
5546         | 'n_cs_precedes' | 'n_sep_by_space'
5547         | 'p_sign_posn' | 'n_sign_posn'
5548         | 'int_p_cs_precedes' | 'int_p_sep_by_space'
5549         | 'int_n_cs_precedes' | 'int_n_sep_by_space'
5550         | 'int_p_sign_posn' | 'int_n_sign_posn'
5551         ;
5552     mon_keyword_grouping : 'mon_grouping'
5553         ;
5554     mon_group_list : NUMBER
5555         | mon_group_list ';' NUMBER
5556         ;
5557     monetary_tlr : 'END' 'LC_MONETARY' EOL
5558         ;
5559     /* The following is the LC_NUMERIC category grammar */
5560     lc_numeric : numeric_hdr numeric_keywords numeric_tlr
5561         | numeric_hdr 'copy' locale_name EOL numeric_tlr
5562         ;
5563     numeric_hdr : 'LC_NUMERIC' EOL
5564         ;
5565     numeric_keywords : numeric_keywords numeric_keyword
5566         | numeric_keyword
5567         ;
5568     numeric_keyword : num_keyword_string num_string EOL
5569         | num_keyword_grouping num_group_list EOL
5570         ;
5571     num_keyword_string : 'decimal_point'
5572         | 'thousands_sep'
5573         ;
5574     num_string : '"' char_list '"'
5575         | '""'
5576         ;
5577     num_keyword_grouping : 'grouping'
5578         ;
5579     num_group_list : NUMBER
5580         | num_group_list ';' NUMBER
5581         ;
5582     numeric_tlr : 'END' 'LC_NUMERIC' EOL
5583         ;
5584     /* The following is the LC_TIME category grammar */
5585     lc_time : time_hdr time_keywords time_tlr
5586         | time_hdr 'copy' locale_name EOL time_tlr

```

```
5587                                     ;
5588     time_hdr                          : 'LC_TIME' EOL
5589                                     ;
5590     time_keywords                       : time_keywords time_keyword
5591     | time_keyword
5592                                     ;
5593     time_keyword                        : time_keyword_name time_list EOL
5594     | time_keyword_fmt time_string EOL
5595     | time_keyword_opt time_list EOL
5596                                     ;
5597     time_keyword_name                   : 'abday' | 'day' | 'abmon' | 'mon'
5598                                     ;
5599     time_keyword_fmt                    : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5600     | 'am_pm' | 't_fmt_ampm'
5601                                     ;
5602     time_keyword_opt                    : 'era' | 'era_d_fmt' | 'era_t_fmt'
5603     | 'era_d_t_fmt' | 'alt_digits'
5604                                     ;
5605     time_list                           : time_list ';' time_string
5606     | time_string
5607                                     ;
5608     time_string                         : '"' char_list '"'
5609                                     ;
5610     time_tlr                            : 'END' 'LC_TIME' EOL
5611                                     ;
```

Environment Variables

8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of POSIX.1-2017 for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain the character '='. For values to be portable across systems conforming to POSIX.1-2017, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in an environment of a process has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of POSIX.1-2017 consist solely of uppercase letters, digits, and the <underscore> ('_') from the characters defined in Table 6-1 (on page 125) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

Note: Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, assigning a new value to the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of POSIX.1-2017.

If the application modifies the pointers to which *environ* points, the behavior of all interfaces described in the System Interfaces volume of POSIX.1-2017 is undefined.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

5650	ARFLAGS	IFS	MAILPATH	PS1
5651	CC	LANG	MAILRC	PS2
5652	CDPATH	LC_ALL	MAKEFLAGS	PS3
5653	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5654	CHARSET	LC_CTYPE	MANPATH	PWD
5655	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5656	DATMSK	LC_MONETARY	MORE	SECONDS
5657	DEAD	LC_NUMERIC	MSGVERB	SHELL
5658	EDITOR	LC_TIME	NLSPATH	TERM
5659	ENV	LDFLAGS	NPROC	TERMCAP
5660	EXINIT	LEX	OLDPWD	TERMINFO
5661	FC	LFLAGS	OPTARG	TMPDIR
5662	FCEDIT	LINENO	OPTERR	TZ
5663	FFLAGS	LINES	OPTIND	USER
5664	GET	LISTER	PAGER	VISUAL
5665	GFLAGS	LOGNAME	PATH	YACC
5666	HISTFILE	LPDEST	PPID	YFLAGS
5667	HISTORY	MAIL	PRINTER	
5668	HISTSIZE	MAILCHECK	PROCLANG	
5669	HOME	MAILER	PROJECTDIR	

5670 If the variables in the following two sections are present in the environment during the
5671 execution of an application or utility, they shall be given the meaning described below. Some are
5672 placed into the environment by the implementation at the time the user logs in; all can be added
5673 or changed by the user or any ancestor of the current process. The implementation adds or
5674 changes environment variables named in POSIX.1-2017 only as specified in POSIX.1-2017. If
5675 they are defined in the application's environment, the utilities in the Shell and Utilities volume
5676 of POSIX.1-2017 and the functions in the System Interfaces volume of POSIX.1-2017 assume they
5677 have the specified meaning. Conforming applications shall not set these environment variables
5678 to have meanings other than as described. See *getenv()* (on page 1029) and XCU [Section 2.12](#) (on
5679 page 2381) for methods of accessing these variables.

5680 8.2 Internationalization Variables

5681 This section describes environment variables that are relevant to the operation of
5682 internationalized interfaces described in POSIX.1-2017.

5683 Users may use the following environment variables to announce specific localization
5684 requirements to applications. Applications can retrieve this information using the *setlocale()*
5685 function to initialize the correct behavior of the internationalized interfaces. The descriptions of
5686 the internationalization environment variables describe the resulting behavior only when the
5687 application locale is initialized in this way. The use of the internationalization variables by
5688 utilities described in the Shell and Utilities volume of POSIX.1-2017 is described in the
5689 ENVIRONMENT VARIABLES section for those utilities in addition to the global effects
5690 described in this section.

5691 **LANG** This variable shall determine the locale category for native language, local
5692 customs, and coded character set in the absence of the *LC_ALL* and other *LC_**
5693 (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*,
5694 *LC_TIME*) environment variables. This can be used by applications to
5695 determine the language to use for error messages and instructions, collating
5696 sequences, date formats, and so on.

5697	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_</i> (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> , <i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) and the <i>LANG</i> environment variable.
5698		
5699		
5700		
5701		
5702	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5703		
5704		
5705		
5706		
5707	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5708		
5709		
5710		
5711		
5712		
5713		
5714	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i> function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by POSIX.1-2017 should be affected by the setting of <i>LC_MESSAGES</i> .
5715		
5716		
5717		
5718		
5719		
5720		
5721		
5722	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.
5723		
5724		
5725	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the string conversion functions in <i>strtod()</i> . Additional semantics of this variable, if any, are implementation-defined.
5726		
5727		
5728		
5729		
5730	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strptime()</i> . Additional semantics of this variable, if any, are implementation-defined.
5731		
5732		
5733	<i>NLSPATH</i>	This variable shall contain a sequence of templates that the <i>catopen()</i> function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a pathname, and an optional suffix.
5734		
5735		
5736		
5737		For example:
5738		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5739		defines that <i>catopen()</i> should look for all message catalogs in the directory /system/nlslib , where the catalog name should be constructed from the <i>name</i> parameter passed to <i>catopen()</i> (<i>%N</i>), with the suffix .cat .
5740		
5741		
5742		Conversion specifications consist of a '%' symbol, followed by a single-letter keyword. The following keywords are currently defined:
5743		

5744 %N The value of the *name* parameter passed to *catopen()*.

5745 %L The value of the *LC_MESSAGES* category.

5746 %l The *language* element from the *LC_MESSAGES* category.

5747 %t The *territory* element from the *LC_MESSAGES* category.

5748 %c The *codeset* element from the *LC_MESSAGES* category.

5749 %% A single '%' character.

5750 An empty string is substituted if the specified value is not currently defined.
5751 The separators <underscore> ('_') and <period> ('.') are not included in
5752 the %t and %c conversion specifications.

5753 Templates defined in *NLSPATH* are separated by <colon> characters (':'). A
5754 leading or two adjacent <colon> characters ("::") is equivalent to specifying
5755 %N. For example:

5756 NLSPATH=":%N.cat:/nlslib/%L/%N.cat"

5757 indicates to *catopen()* that it should look for the requested message catalog in
5758 *name*, *name.cat*, and */nlslib/category/name.cat*, where *category* is the value of the
5759 *LC_MESSAGES* category of the current locale.

5760 Users should not set the *NLSPATH* variable unless they have a specific reason
5761 to override the default system path. Setting *NLSPATH* to override the default
5762 system path produces undefined results in the standard utilities and in
5763 applications with appropriate privileges.

5764 The environment variables *LANG*, *LC_ALL*, *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*,
5765 *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*, and *NLSPATH* provide for the support of
5766 internationalized applications. The standard utilities shall make use of these environment
5767 variables as described in this section and the individual ENVIRONMENT VARIABLES sections
5768 for the utilities. If these variables specify locale categories that are not based upon the same
5769 underlying codeset, the results are unspecified.

5770 The values of locale categories shall be determined by a precedence order; the first condition met
5771 below determines the value:

- 5772 1. If the *LC_ALL* environment variable is defined and is not null, the value of *LC_ALL* shall
5773 be used.
- 5774 2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*,
5775 *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the
5776 environment variable shall be used to initialize the category that corresponds to the
5777 environment variable.
- 5778 3. If the *LANG* environment variable is defined and is not null, the value of the *LANG*
5779 environment variable shall be used.
- 5780 4. If the *LANG* environment variable is not set or is set to the empty string, the
5781 implementation-defined default locale shall be used.

5782 If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
5783 behave in accordance with the rules in [Section 7.2](#) (on page 136) for the associated category.

5784 If the locale value begins with a <slash>, it shall be interpreted as the pathname of a file that was
5785 created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*.
5786 Referencing such a pathname shall result in that locale being used for the indicated category.

5787 XSI If the locale value has the form:

5788 `language[_territory] [.codeset]`

5789 it refers to an implementation-provided locale, where settings of language, territory, and codeset
5790 are implementation-defined.

5791 `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are
5792 defined to accept an additional field `@modifier`, which allows the user to select a specific instance
5793 of localization data within a single category (for example, for selecting the dictionary as opposed
5794 to the character ordering of data). The syntax for these environment variables is thus defined as:

5795 `[language[_territory] [.codeset] [@modifier]]`

5796 For example, if a user wanted to interact with the system in French, but required to sort German
5797 text files, `LANG` and `LC_COLLATE` could be defined as:

5798 `LANG=Fr_FR`
5799 `LC_COLLATE=De_DE`

5800 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for
5801 example:

5802 `LC_COLLATE=De_DE@dict`

5803 An implementation may support other formats.

5804 If the locale value is not recognized by the implementation, the behavior is unspecified.

5805 These environment variables are used by the `newlocale()` and `setlocale()` functions, and by the
5806 standard utilities.

5807 Additional criteria for determining a valid locale name are implementation-defined.

5808 8.3 Other Environment Variables

5809 `COLUMNS` This variable shall represent a decimal integer >0 used to indicate the user's
5810 preferred width in column positions for the terminal screen or window; see
5811 [Section 3.103](#) (on page 50). If this variable is unset or null, the implementation
5812 determines the number of columns, appropriate for the terminal or window,
5813 in an unspecified manner. When `COLUMNS` is set, any terminal-width
5814 information implied by `TERM` is overridden. Users and conforming
5815 applications should not set `COLUMNS` unless they wish to override the
5816 system selection and produce output unrelated to the terminal characteristics.

5817 Users should not need to set this variable in the environment unless there is a
5818 specific reason to override the implementation's default behavior, such as to
5819 display data in an area arbitrarily smaller than the terminal or window.

5820 XSI `DATEMSK` Indicates the pathname of the template file used by `getdate()`.

5821 `HOME` The system shall initialize this variable at the time of login to be a pathname of
5822 the user's home directory. See [<pwd.h>](#).

5823 `LINES` This variable shall represent a decimal integer >0 used to indicate the user's
5824 preferred number of lines on a page or the vertical screen or window size in
5825 lines. A line in this case is a vertical measure large enough to hold the tallest
5826 character in the character set being displayed. If this variable is unset or null,

5827		the implementation determines the number of lines, appropriate for the
5828		terminal or window (size, terminal baud rate, and so on), in an unspecified
5829		manner. When <i>LINES</i> is set, any terminal-height information implied by
5830		<i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i>
5831		unless they wish to override the system selection and produce output
5832		unrelated to the terminal characteristics.
5833		Users should not need to set this variable in the environment unless there is a
5834		specific reason to override the implementation's default behavior, such as to
5835		display data in an area arbitrarily smaller than the terminal or window.
5836	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's
5837		login name. See <pwd.h> . For a value of <i>LOGNAME</i> to be portable across
5838		implementations of POSIX.1-2017, the value should be composed of characters
5839		from the portable filename character set.
5840	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by
5841		<i>fntmsg()</i> .
5842	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain
5843		functions and utilities apply in searching for an executable file known only by
5844		a filename. The prefixes shall be separated by a <colon> (':'). When a non-
5845		zero-length prefix is applied to this filename, a <slash> shall be inserted
5846		between the prefix and the filename if the prefix did not end in <slash>. A
5847		zero-length prefix is a legacy feature that indicates the current working
5848		directory. It appears as two adjacent <colon> characters ("::"), as an initial
5849		<colon> preceding the rest of the list, or as a trailing <colon> following the
5850		rest of the list. A strictly conforming application shall use an actual pathname
5851		(such as <i>.</i>) to represent the current working directory in <i>PATH</i> . The list shall
5852		be searched from beginning to end, applying the filename to each prefix, until
5853		an executable file with the specified name and appropriate execution
5854		permissions is found. If the pathname being sought contains a <slash>, the
5855		search through the path prefixes shall not be performed. If the pathname
5856		begins with a <slash>, the specified path is resolved (see Section 4.13 , on page
5857		111). If <i>PATH</i> is unset or is set to null, the path search is implementation-
5858		defined.
5859		Since <colon> is a separator in this context, directory names that might be
5860		used in <i>PATH</i> should not include a <colon> character.
5861	<i>PWD</i>	This variable shall represent an absolute pathname of the current working
5862		directory. It shall not contain any components that are dot or dot-dot. The
5863		value is set by the <i>cd</i> utility, and by the <i>sh</i> utility during initialization.
5864	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command
5865		language interpreter. If this interpreter does not conform to the Shell
5866		Command Language in XCU Chapter 2 (on page 2345), utilities may behave
5867		differently from those described in POSIX.1-2017.
5868	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for
5869		programs that need a place to create temporary files.
5870	<i>TERM</i>	This variable shall represent the terminal type for which output is to be
5871		prepared. This information is used by utilities and application programs
5872		wishing to exploit special capabilities specific to a terminal. The format and
5873		allowable values of this environment variable are unspecified.

5874	<i>TZ</i>	This variable shall represent timezone information. The contents of the environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>ctime_r()</i> , <i>localtime()</i> , <i>localtime_r()</i> , <i>strptime()</i> , <i>mktime()</i> , functions, and by various utilities, to override the default timezone. The value of <i>TZ</i> has one of the two forms (spaces inserted for clarity):
5875		
5876		
5877		
5878		
5879		<i>:characters</i>
5880		or:
5881		<i>std offset dst offset, rule</i>
5882		If <i>TZ</i> is of the first format (that is, if the first character is a <colon>), the characters following the <colon> are handled in an implementation-defined manner.
5883		
5884		
5885		The expanded format (for all <i>TZ</i> s whose value does not have a <colon> as the first character) is as follows:
5886		
5887		<i>stdoffset[dst[offset][,start[/time],end[/time]]]</i>
5888		Where:
5889	<i>std</i> and <i>dst</i>	Indicate no less than three, nor more than {TZNAME_MAX}, bytes that are the designation for the standard (<i>std</i>) or the alternative (<i>dst</i> ‡such as Daylight Savings Time) timezone. Only <i>std</i> is required; if <i>dst</i> is missing, then the alternative time does not apply in this locale.
5890		
5891		
5892		
5893		
5894		Each of these fields may occur in either of two formats quoted or unquoted:
5895		
5896		In the quoted form, the first character shall be the <less-than-sign> ('<') character and the last character shall be the <greater-than-sign> ('>') character. All characters between these quoting characters shall be alphanumeric characters from the portable character set in the current locale, the <plus-sign> ('+') character, or the <hyphen-minus> ('-') character. The <i>std</i> and <i>dst</i> fields in this case shall not include the quoting characters.
5897		
5898		
5899		
5900		
5901		
5902		
5903		
5904		In the unquoted form, all characters in these fields shall be alphabetic characters from the portable character set in the current locale.
5905		
5906		
5907		The interpretation of these fields is unspecified if either field is less than three bytes (except for the case when <i>dst</i> is missing), more than {TZNAME_MAX} bytes, or if they contain characters other than those specified.
5908		
5909		
5910		
5911	<i>offset</i>	Indicates the value added to the local time to arrive at Coordinated Universal Time. The <i>offset</i> has the form:
5912		
5913		<i>hh[:mm[:ss]]</i>
5914		The minutes (<i>mm</i>) and seconds (<i>ss</i>) are optional. The hour (<i>hh</i>) shall be required and may be a single digit. The <i>offset</i> following <i>std</i> shall be required. If no <i>offset</i> follows <i>dst</i> , the alternative time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal
5915		
5916		
5917		
5918		

5919 number. The hour shall be between zero and 24, and the minutes
 5920 (and seconds)—if present—between zero and 59. The result of
 5921 using values outside of this range is unspecified. If preceded by
 5922 a '-', the timezone shall be east of the Prime Meridian;
 5923 otherwise, it shall be west (which may be indicated by an
 5924 optional preceding '+').

5925 *rule* Indicates when to change to and back from the alternative time.
 5926 The *rule* has the form:

5927 `date[/time], date[/time]`

5928 where the first *date* describes when the change from standard to
 5929 alternative time occurs and the second *date* describes when the
 5930 change back happens. Each *time* field describes when, in current
 5931 local time, the change to the other time is made.

5932 The format of *date* is one of the following:

5933 *Jn* The Julian day n ($1 \leq n \leq 365$). Leap days shall not be
 5934 counted. That is, in all years ‡including leap years ‡
 5935 February 28 is day 59 and March 1 is day 60. It is
 5936 impossible to refer explicitly to the occasional February
 5937 29.

5938 *n* The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
 5939 be counted, and it is possible to refer to February 29.

5940 *Mm.n.d* The d 'th day ($0 \leq d \leq 6$) of week n of month m of the
 5941 year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means "the last
 5942 d day in month m " which may occur in either the fourth
 5943 or the fifth week). Week 1 is the first week in which the
 5944 d 'th day occurs. Day zero is Sunday.

5945 The *time* has the same format as *offset* except that no leading sign
 5946 ('-' or '+') is allowed. The default, if *time* is not given, shall be
 5947 02:00:00.

Regular Expressions

5950 Regular Expressions (REs) provide a mechanism to select specific strings from a set of character
5951 strings.

5952 Regular expressions are a context-independent syntax that can represent a wide variety of
5953 character sets and character set orderings, where these character sets are interpreted according
5954 to the current locale. While many regular expressions can be interpreted differently depending
5955 on the current locale, many features, such as character class expressions, provide for contextual
5956 invariance across locales.

5957 The Basic Regular Expression (BRE) notation and construction rules in [Section 9.3](#) (on page 183)
5958 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the
5959 Extended Regular Expressions (ERE) described in [Section 9.4](#) (on page 188); any exceptions for
5960 both cases are noted in the descriptions of the specific utilities using regular expressions. Both
5961 BREs and EREs are supported by the Regular Expression Matching interface in the System
5962 Interfaces volume of POSIX.1-2017 under *regcomp()*, *regexexec()*, and related functions.

5963 9.1 Regular Expression Definitions

5964 For the purposes of this section, the following definitions shall apply:

5965 **entire regular expression**

5966 The concatenated set of one or more BREs or EREs that make up the pattern specified for
5967 string selection.

5968 **matched**

5969 A sequence of zero or more characters shall be said to be matched by a BRE or ERE when
5970 the characters in the sequence correspond to a sequence of characters defined by the
5971 pattern.

5972 Matching shall be based on the bit pattern used for encoding the character, not on the
5973 graphic representation of the character. This means that if a character set contains two or
5974 more encodings for a graphic symbol, or if the strings searched contain text encoded in
5975 more than one codeset, no attempt is made to search for any other representation of the
5976 encoded symbol. If that is required, the user can specify equivalence classes containing all
5977 variations of the desired graphic symbol.

5978 The search for a matching sequence starts at the beginning of a string and stops when the
5979 first sequence matching the expression is found, where “first” is defined to mean “begins
5980 earliest in the string”. If the pattern permits a variable number of matching characters and
5981 thus there is more than one such sequence starting at that point, the longest such sequence
5982 is matched. For example, the BRE “bb*” matches the second to fourth characters of the
5983 string “abbbc”, and the ERE “(wee|week)(knights|night)” matches all ten
5984 characters of the string “weeknights”.

5985 Consistent with the whole match being the longest of the leftmost matches, each subpattern,
5986 from left to right, shall match the longest possible string. For this purpose, a null string shall
5987 be considered to be longer than no match at all. For example, matching the BRE

5988 "\(.*\).*" against "abcdef", the subexpression "(\\1)" is "abcdef", and matching
5989 the BRE "(a*\\)*" against "bc", the subexpression "(\\1)" is the null string.

5990 When a multi-character collating element in a bracket expression (see [Section 9.3.5](#), on page
5991 184) is involved, the longest sequence shall be measured in characters consumed from the
5992 string to be matched; that is, the collating element counts not as one element, but as the
5993 number of characters it matches.

5994 **BRE (ERE) matching a single character**

5995 A BRE or ERE that shall match either a single character or a single collating element.

5996 Only a BRE or ERE of this type that includes a bracket expression (see [Section 9.3.5](#), on page
5997 184) can match a collating element.

5998 **BRE (ERE) matching multiple characters**

5999 A BRE or ERE that shall match a concatenation of single characters or collating elements.

6000 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE
6001 (ERE) special characters.

6002 **invalid**

6003 This section uses the term “invalid” for certain constructs or conditions. Invalid REs shall
6004 cause the utility or function using the RE to generate an error condition. When invalid is not
6005 used, violations of the specified syntax or semantics for REs produce undefined results: this
6006 may entail an error, enabling an extended syntax for that RE, or using the construct in error
6007 as literal characters to be matched. For example, the BRE construct "\\{1,2,3\\}" does not
6008 comply with the grammar. A conforming application cannot rely on it producing an error
6009 nor matching the literal characters "\\{1,2,3\\}”.

6010 **9.2 Regular Expression General Requirements**

6011 The requirements in this section shall apply to both basic and extended regular expressions.

6012 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)
6013 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter
6014 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that
6015 is, zero or more characters followed by a <newline>.

6016 In the functions processing regular expressions described in System Interfaces volume of
6017 POSIX.1-2017, the <newline> is regarded as an ordinary character and both a <period> and a
6018 non-matching list can match one. The Shell and Utilities volume of POSIX.1-2017 specifies
6019 within the individual descriptions of those standard utilities employing regular expressions
6020 whether they permit matching of <newline> characters; if not stated otherwise, the use of literal
6021 <newline> characters or any escape sequence equivalent in either patterns or matched text
6022 produces undefined results. Those utilities (like *grep*) that do not allow <newline> characters to
6023 match are responsible for eliminating any <newline> from strings before matching against the
6024 RE. The *regcomp()* function in the System Interfaces volume of POSIX.1-2017, however, can
6025 provide support for such processing without violating the rules of this section.

6026 The interfaces specified in POSIX.1-2017 do not permit the inclusion of a NUL character in an RE
6027 or in the string to be matched. If during the operation of a standard utility a NUL is included in
6028 the text designated to be matched, that NUL may designate the end of the text string for the
6029 purposes of matching.

6030 When a standard utility or function that uses regular expressions specifies that pattern matching
6031 shall be performed without regard to the case (uppercase or lowercase) of either data or

6032 patterns, then when each character in the string is matched against the pattern, not only the
 6033 character, but also its case counterpart (if any), shall be matched. This definition of case-
 6034 insensitive processing is intended to allow matching of multi-character collating elements as
 6035 well as characters, as each character in the string is matched using both its cases. For example, in
 6036 a locale where "Ch" is a multi-character collating element and where a matching list expression
 6037 matches such elements, the RE "[[.Ch.]]" when matched against the string "char" is in
 6038 reality matched against "ch", "Ch", "cH", and "CH".

6039 The implementation shall support any regular expression that does not exceed 256 bytes in
 6040 length.

6041 9.3 Basic Regular Expressions

6042 9.3.1 BREs Matching a Single Character or Collating Element

6043 A BRE ordinary character, a special character preceded by a <backslash>, or a <period> shall
 6044 match a single character. A bracket expression shall match a single character or a single collating
 6045 element.

6046 9.3.2 BRE Ordinary Characters

6047 An ordinary character is a BRE that matches itself: any character in the supported character set,
 6048 except for the BRE special characters listed in [Section 9.3.3](#).

6049 The interpretation of an ordinary character preceded by an unescaped <backslash> ('\\') is
 6050 undefined, except for:

6051 The characters ')', '(', '{', and '}'

6052 The digits 1 to 9 inclusive (see [Section 9.3.6](#), on page 186)

6053 A character inside a bracket expression

6054 9.3.3 BRE Special Characters

6055 A BRE special character has special properties in certain contexts. Outside those contexts, or
 6056 when preceded by a <backslash>, such a character is a BRE that matches the special character
 6057 itself. The BRE special characters and the contexts in which they have their special meaning are
 6058 as follows:

6059 . [\ The <period>, <left-square-bracket>, and <backslash> shall be special except when
 6060 used in a bracket expression (see [Section 9.3.5](#), on page 184). An expression containing
 6061 a '[' that is unescaped and is not part of a bracket expression produces undefined
 6062 results.

6063 * The <asterisk> shall be special except when used:

6064 p̄n'albracket expression

- 6065 ‡ \$ The first character of an entire BRE (after an initial '^', if any)
- 6066 ‡ \$ The first character of a subexpression (after an initial '^', if any); see [Section](#)
- 6067 [9.3.6](#) (on page 186)
- 6068 ^ The <circumflex> shall be special when used as an anchor (see [Section 9.3.8](#), on page
- 6069 188). The <circumflex> shall signify a non-matching list expression when it occurs first
- 6070 in a list, immediately following a <left-square-bracket> (see [Section 9.3.5](#)).
- 6071 \$ The <dollar-sign> shall be special when used as an anchor.

6072 9.3.4 Periods in BREs

6073 A <period> ('.'), when used outside a bracket expression, is a BRE that shall match any

6074 character in the supported character set except NUL.

6075 9.3.5 RE Bracket Expression

6076 A bracket expression (an expression enclosed in square brackets, "[]") is an RE that shall

6077 match a specific set of single characters, and may match a specific set of multi-character collating

6078 elements, based on the non-empty set of list expressions contained in the bracket expression.

6079 The following rules and definitions apply to bracket expressions:

- 6080 1. A bracket expression is either a matching list expression or a non-matching list
- 6081 expression. It consists of one or more expressions: ordinary characters, collating elements,
- 6082 collating symbols, equivalence classes, character classes, or range expressions. The <right-
- 6083 square-bracket> (']') shall lose its special meaning and represent itself in a bracket
- 6084 expression if it occurs first in the list (after an initial <circumflex> ('^'), if any).
- 6085 Otherwise, it shall terminate the bracket expression, unless it appears in a collating
- 6086 symbol (such as "[.].]") or is the ending <right-square-bracket> for a collating symbol,
- 6087 equivalence class, or character class. The special characters '.', '*', '[', and '\\'
- 6088 (<period>, <asterisk>, <left-square-bracket>, and <backslash>, respectively) shall lose
- 6089 their special meaning within a bracket expression.

6090 The character sequences "[.", "[=", and "[:" (<left-square-bracket> followed by a

6091 <period>, <equals-sign>, or <colon>) shall be special inside a bracket expression and are

6092 used to delimit collating symbols, equivalence class expressions, and character class

6093 expressions. These symbols shall be followed by a valid expression and the matching

6094 terminating sequence ".]", "=]", or ":]", as described in the following items.

- 6095 2. A matching list expression specifies a list that shall match any single character that is
- 6096 matched by one of the expressions represented in the list. The first character in the list
- 6097 cannot be the <circumflex>. An ordinary character in the list should only match that
- 6098 character, but may match any single character that collates equally with that character; for
- 6099 example, "[abc]" is an RE that should only match one of the characters 'a', 'b', or
- 6100 'c'.

6101 **Note:** A future version of this standard may require that an ordinary character in the list only

6102 matches that character.

6103 It is unspecified whether a matching list expression matches a multi-character collating

6104 element that is matched by one of the expressions.

- 6105 3. A non-matching list expression begins with a <circumflex> ('^'), and the matching
- 6106 behavior shall be the logical inverse of the corresponding matching list expression (the
- 6107 same bracket expression but without the leading <circumflex>). For example, if the RE

6108 "[abc]" only matches 'a', 'b', or 'c', then "[^abc]" is an RE that matches any
 6109 character except 'a', 'b', or 'c'. It is unspecified whether a non-matching list
 6110 expression matches a multi-character collating element that is not matched by any of the
 6111 expressions. The <circumflex> shall have this special meaning only when it occurs first in
 6112 the list, immediately following the <left-square-bracket>.

6113 4. A collating symbol is a collating element enclosed within bracket-period ("[" and
 6114 ".]") delimiters. Collating elements are defined as described in [Section 7.3.2.4](#) (on page
 6115 149). Conforming applications shall represent multi-character collating elements as
 6116 collating symbols when it is necessary to distinguish them from a list of the individual
 6117 characters that make up the multi-character collating element. For example, if the string
 6118 "ch" is a collating element defined using the line:

```
6119 collating-element <ch-digraph> from "<c><h>"
```

6120 in the locale definition, the expression "[.ch.]" shall be treated as an RE containing
 6121 the collating symbol 'ch', while "[ch]" shall be treated as an RE matching 'c' or 'h'.
 6122 Collating symbols are recognized only inside bracket expressions. If the string is not a
 6123 collating element in the current locale, the expression is invalid.

6124 5. An equivalence class expression shall represent the set of collating elements belonging to
 6125 an equivalence class, as described in [Section 7.3.2.4](#) (on page 149). Only primary
 6126 equivalence classes shall be recognized. The class shall be expressed by enclosing any one
 6127 of the collating elements in the equivalence class within bracket-equal ("[" and "=")
 6128 delimiters. For example, if 'a', ' ', and '^' belong to the same equivalence class, then
 6129 "[[=a=]b]", "[[==]b]", and "[[=^=]b]" are each equivalent to "[a^b]". If the
 6130 collating element does not belong to an equivalence class, the equivalence class
 6131 expression shall be treated as a collating symbol.

6132 6. A character class expression shall represent the union of two sets:

6133 a. The set of single characters that belong to the character class, as defined in the
 6134 *LC_CTYPE* category in the current locale.

6135 b. An unspecified set of multi-character collating elements.

6136 All character classes specified in the current locale shall be recognized. A character class
 6137 expression is expressed as a character class name enclosed within bracket-<colon> ("[" : "
 6138 and " :]") delimiters.

6139 The following character class expressions shall be supported in all locales:

```
6140 [:alnum:] [:cntrl:] [:lower:] [:space:]
6141 [:alpha:] [:digit:] [:print:] [:upper:]
6142 [:blank:] [:graph:] [:punct:] [:xdigit:]
```

6143 In addition, character class expressions of the form:

```
6144 [:name:]
```

6145 are recognized in those locales where the *name* keyword has been given a **charclass**
 6146 definition in the *LC_CTYPE* category.

6147 7. In the POSIX locale, a range expression represents the set of collating elements that fall
 6148 between two elements in the collation sequence, inclusive. In other locales, a range
 6149 expression has unspecified behavior: strictly conforming applications shall not rely on
 6150 whether the range expression is valid, or on the set of collating elements matched. A
 6151 range expression shall be expressed as the starting point and the ending point separated
 6152 by a <hyphen-minus> ('-').

- 6153 In the following, all examples assume the POSIX locale.
- 6154 The starting range point and the ending range point shall be a collating element or
6155 collating symbol. An equivalence class expression used as a starting or ending point of a
6156 range expression produces unspecified results. An equivalence class can be used portably
6157 within a bracket expression, but only outside the range. If the represented set of collating
6158 elements is empty, it is unspecified whether the expression matches nothing, or is treated
6159 as invalid.
- 6160 The interpretation of range expressions where the ending range point is also the starting
6161 range point of a subsequent range expression (for example, "[a-m-o]") is undefined.
- 6162 The <hyphen-minus> character shall be treated as itself if it occurs first (after an initial
6163 '^', if any) or last in the list, or as an ending range point in a range expression. As
6164 examples, the expressions "[-ac]" and "[ac -]" are equivalent and match any of the
6165 characters 'a', 'c', or '-'; "[^-ac]" and "[^ac -]" are equivalent and match any
6166 characters except 'a', 'c', or '-'; the expression "[%--]" matches any of the
6167 characters between '%' and '-' inclusive; the expression "[--@]" matches any of the
6168 characters between '-' and '@' inclusive; and the expression "[a--@]" is either invalid
6169 or equivalent to '@', because the letter 'a' follows the symbol '-' in the POSIX locale.
6170 To use a <hyphen-minus> as the starting range point, it shall either come first in the
6171 bracket expression or be specified as a collating symbol; for example, "[] [. - .] - 0]",
6172 which matches either a <right-square-bracket> or any character or collating element that
6173 collates between <hyphen-minus> and 0, inclusive.
- 6174 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the
6175 '^', if any) and the '-' last within the bracket expression.
- 6176 8. If a bracket expression contains at least three list elements, where the first and last list
6177 elements are the same single-character element of <period>, <equals-sign>, or <colon>,
6178 then it is unspecified whether the bracket expression will be treated as a collating symbol,
6179 equivalence class, or character class, respectively; treated as a matching list expression; or
6180 rejected as an error.

6181 9.3.6 BREs Matching Multiple Characters

- 6182 The following rules can be used to construct BREs matching multiple characters from BREs
6183 matching a single character:
- 6184 1. The concatenation of BREs shall match the concatenation of the strings matched by each
6185 component of the BRE.
 - 6186 2. A subexpression can be defined within a BRE by enclosing it between the character pairs
6187 "\(" and "\)". Such a subexpression shall match whatever it would have matched
6188 without the "\(" and "\)", except that anchoring within subexpressions is optional
6189 behavior; see [Section 9.3.8](#) (on page 188). Subexpressions can be arbitrarily nested.
 - 6190 3. The back-reference expression '\n' shall match the same (possibly empty) string of
6191 characters as was matched by a subexpression enclosed between "\(" and "\)"
6192 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the
6193 *n*th subexpression (the one that begins with the *n*th "\(" from the beginning of the
6194 pattern and ends with the corresponding paired "\)"). The expression is invalid if less
6195 than *n* subexpressions precede the '\n'. The string matched by a contained
6196 subexpression shall be within the string matched by the containing subexpression. If the
6197 containing subexpression does not match, or if there is no match for the contained
6198 subexpression within the string matched by the containing subexpression, then back-

6199 reference expressions corresponding to the contained subexpression shall not match.
 6200 When a subexpression matches more than one string, a back-reference expression
 6201 corresponding to the subexpression shall refer to the last matched string. For example, the
 6202 expression "`^\(.*\)\1$`" matches strings consisting of two adjacent appearances of the
 6203 same substring, and the expression "`\(a\)*\1`" fails to match 'a', the expression
 6204 "`\(a\(b\)*\)*\2`" fails to match 'abab', and the expression "`^\(ab*\)\1$`"
 6205 matches 'ababbabb', but fails to match 'ababbab'.

6206 4. When a BRE matching a single character, a subexpression, or a back-reference is followed
 6207 by the special character <asterisk> ('*'), together with that <asterisk> it shall match
 6208 what zero or more consecutive occurrences of the BRE would match. For example,
 6209 "`[ab]*`" and "`[ab][ab]`" are equivalent when matching the string "ab".

6210 5. When a BRE matching a single character, a subexpression, or a back-reference is followed
 6211 by an interval expression of the format "`\{m\}`", "`\{m,\}`", or "`\{m,n\}`", together
 6212 with that interval expression it shall match what repeated consecutive occurrences of the
 6213 BRE would match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences
 6214 and *n* specifies the maximum number of occurrences. The expression "`\{m\}`" shall
 6215 match exactly *m* occurrences of the preceding BRE, "`\{m,\}`" shall match at least *m*
 6216 occurrences, and "`\{m,n\}`" shall match any number of occurrences between *m* and *n*,
 6217 inclusive.
 6218

6219 For example, in the string "abababcccccd" the BRE "`c\{3\}`" is matched by
 6220 characters seven to nine, the BRE "`\(ab\)\{4,\}`" is not matched at all, and the BRE
 6221 "`c\{1,3\}d`" is matched by characters ten to thirteen.

6222 The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined
 6223 results.

6224 A subexpression repeated by an <asterisk> ('*') or an interval expression shall not match a null
 6225 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or
 6226 minimum number of occurrences for the interval expression.

6227 9.3.7 BRE Precedence

6228 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	[<code>==</code>] [<code>::</code>] [<code>..</code>]
Escaped characters	\<special character>
Bracket expression	[]
Subexpressions/back-references	\(\) \n
Single-character-BRE duplication	* \{m,n\}
Concatenation	
Anchoring	^ \$

6237 9.3.8 BRE Expression Anchoring

6238 A BRE can be limited to matching expressions that begin or end a string; this is called
6239 “anchoring”. The <circumflex> and <dollar-sign> special characters shall be considered BRE
6240 anchors in the following contexts:

- 6241 1. A <circumflex> ('^') shall be an anchor when used as the first character of an entire BRE.
6242 The implementation may treat the <circumflex> as an anchor when used as the first
6243 character of a subexpression. The <circumflex> shall anchor the expression (or optionally
6244 subexpression) to the beginning of a string; only sequences starting at the first character
6245 of a string shall be matched by the BRE. For example, the BRE "^ab" matches "ab" in
6246 the string "abcdef", but fails to match in the string "cdefab". The BRE "\(^ab\)"
6247 may match the former string. A portable BRE shall escape a leading <circumflex> in a
6248 subexpression to match a literal circumflex.
- 6249 2. A <dollar-sign> ('\$') shall be an anchor when used as the last character of an entire BRE.
6250 The implementation may treat a <dollar-sign> as an anchor when used as the last
6251 character of a subexpression. The <dollar-sign> shall anchor the expression (or optionally
6252 subexpression) to the end of the string being matched; the <dollar-sign> can be said to
6253 match the end-of-string following the last character.
- 6254 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the
6255 BRE "^abcdef\$" matches strings consisting only of "abcdef".

6256 9.4 Extended Regular Expressions

6257 The extended regular expression (ERE) notation and construction rules shall apply to utilities
6258 defined as using extended regular expressions; any exceptions to the following rules are noted
6259 in the descriptions of the specific utilities using EREs.

6260 9.4.1 EREs Matching a Single Character or Collating Element

6261 An ERE ordinary character, a special character preceded by a <backslash> or a <period> shall
6262 match a single character. A bracket expression shall match a single character or a single collating
6263 element. An ERE matching a single character enclosed in parentheses shall match the same as
6264 the ERE without parentheses would have matched.

6265 9.4.2 ERE Ordinary Characters

6266 An ordinary character is an ERE that matches itself. An ordinary character is any character in the
6267 supported character set, except for the ERE special characters listed in [Section 9.4.3](#) (on page
6268 189). The interpretation of an ordinary character preceded by an unescaped <backslash> ('\\')
6269 is undefined, except in the context of a bracket expression (see [Section 9.4.5](#), on page 189).

6270 9.4.3 ERE Special Characters

6271 An ERE special character has special properties in certain contexts. Outside those contexts, or
6272 when preceded by a <backslash>, such a character shall be an ERE that matches the special
6273 character itself. The extended regular expression special characters and the contexts in which
6274 they shall have their special meaning are as follows:

6275 . [\ (The <period>, <left-square-bracket>, <backslash>, and <left-parenthesis> shall be
6276 special except when used in a bracket expression (see [Section 9.3.5](#), on page 184).
6277 Outside a bracket expression, a <left-parenthesis> immediately followed by a <right-
6278 parenthesis> produces undefined results. A <left-square-bracket> that is unescaped
6279 and is not part of a bracket expression also produces undefined results.

6280) The <right-parenthesis> shall be special when matched with a preceding <left-
6281 parenthesis>, both outside a bracket expression.

6282 * + ? { The <asterisk>, <plus-sign>, <question-mark>, and <left-brace> shall be special except
6283 when used in a bracket expression (see [Section 9.3.5](#), on page 184). Any of the
6284 following uses produce undefined results:

6285 ‡ These characters appear first in an ERE, or immediately following an unescaped
6286 <vertical-line>, <circumflex>, <dollar-sign>, or <left-parenthesis>

6287 ‡ †<left-brace> is not part of a valid interval expression (see [Section 9.4.6](#), on
6288 page 190)

6289 | The <vertical-line> is special except when used in a bracket expression (see [Section](#)
6290 [9.3.5](#), on page 184). A <vertical-line> appearing first or last in an ERE, or immediately
6291 following a <vertical-line> or a <left-parenthesis>, or immediately preceding a <right-
6292 parenthesis>, produces undefined results.

6293 ^ The <circumflex> shall be special when used as an anchor (see [Section 9.4.9](#), on page
6294 191). The <circumflex> shall signify a non-matching list expression when it occurs first
6295 in a list, immediately following a <left-square-bracket> (see [Section 9.3.5](#), on page 184).

6296 \$ The <dollar-sign> shall be special when used as an anchor.

6297 9.4.4 Periods in EREs

6298 A <period> ('.'), when used outside a bracket expression, is an ERE that shall match any
6299 character in the supported character set except NUL.

6300 9.4.5 ERE Bracket Expression

6301 The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see [Section](#)
6302 [9.3.5](#) (on page 184).

6303 9.4.6 EREs Matching Multiple Characters

6304 The following rules shall be used to construct EREs matching multiple characters from EREs
6305 matching a single character:

- 6306 1. A concatenation of EREs shall match the concatenation of the character sequences
6307 matched by each component of the ERE. A concatenation of EREs enclosed in parentheses
6308 shall match whatever the concatenation without the parentheses matches. For example,
6309 both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of
6310 the string "abcdefabcdef".
- 6311 2. When an ERE matching a single character or an ERE enclosed in parentheses is followed
6312 by the special character <plus-sign> ('+'), together with that <plus-sign> it shall match
6313 what one or more consecutive occurrences of the ERE would match. For example, the
6314 ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde".
6315 And, "[ab]+" and "[ab][ab]*" are equivalent.
- 6316 3. When an ERE matching a single character or an ERE enclosed in parentheses is followed
6317 by the special character <asterisk> ('*'), together with that <asterisk> it shall match
6318 what zero or more consecutive occurrences of the ERE would match. For example, the
6319 ERE "b*c" matches the first character in the string "cabbbbcde", and the ERE "b*cd"
6320 matches the third to seventh characters in the string "cabbbbcdebbbbbbcbcd". And,
6321 "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".
- 6322 4. When an ERE matching a single character or an ERE enclosed in parentheses is followed
6323 by the special character <question-mark> ('?'), together with that <question-mark> it
6324 shall match what zero or one consecutive occurrences of the ERE would match. For
6325 example, the ERE "b?c" matches the second character in the string "acabbbbcde".
- 6326 5. When an ERE matching a single character or an ERE enclosed in parentheses is followed
6327 by an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that
6328 interval expression it shall match what repeated consecutive occurrences of the ERE
6329 would match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences
6330 and *n* specifies the maximum number of occurrences. The expression "{m}" matches
6331 exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and
6332 "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.
6333
6334 For example, in the string "abababcccccccd" the ERE "c{3}" is matched by characters
6335 seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

6336 The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces
6337 undefined results.

6338 An ERE matching a single character repeated by an '*', '?', or an interval expression shall not
6339 match a null expression unless this is the only match for the repetition or it is necessary to satisfy
6340 the exact or minimum number of occurrences for the interval expression.

6341 **9.4.7 ERE Alternation**

6342 Two EREs separated by the special character <vertical-line> ('|') shall match a string that is
 6343 matched by either. For example, the ERE "a((bc)|d)" matches the string "abc" and the
 6344 string "ad". Single characters, or expressions matching single characters, separated by the
 6345 <vertical-line> and enclosed in parentheses, shall be treated as an ERE matching a single
 6346 character.

6347 **9.4.8 ERE Precedence**

6348 The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
6349 Collation-related bracket symbols	[==] [::] [..]
6350 Escaped characters	\<special character>
6351 Bracket expression	[]
6352 Grouping	()
6353 Single-character-ERE duplication	* + ? {m,n}
6354 Concatenation	
6355 Anchoring	^ \$
6356 Alternation	
6357	

6358 For example, the ERE "abba|cde" matches either the string "abba" or the string "cde"
 6359 (rather than the string "abbade" or "abbcde", because concatenation has a higher order of
 6360 precedence than alternation).

6361 **9.4.9 ERE Expression Anchoring**

6362 An ERE can be limited to matching expressions that begin or end a string; this is called
 6363 "anchoring". The <circumflex> and <dollar-sign> special characters shall be considered ERE
 6364 anchors when used anywhere outside a bracket expression. This shall have the following effects:

- 6365 1. A <circumflex> ('^') outside a bracket expression shall anchor the expression or
 6366 subexpression it begins to the beginning of a string; such an expression or subexpression
 6367 can match only a sequence starting at the first character of a string. For example, the EREs
 6368 "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string
 6369 "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents
 6370 the expression "a^b" from matching starting at the first character.
- 6371 2. A <dollar-sign> ('\$') outside a bracket expression shall anchor the expression or
 6372 subexpression it ends to the end of a string; such an expression or subexpression can
 6373 match only a sequence ending at the last character of a string. For example, the EREs
 6374 "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string
 6375 "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents
 6376 the expression "e\$f" from matching ending at the last character.

6377 9.5 Regular Expression Grammar

6378 Grammars describing the syntax of both basic and extended regular expressions are presented in
6379 this section. The grammar takes precedence over the text. See XCU [Section 1.3](#) (on page 2335).

6380 9.5.1 BRE/ERE Grammar Lexical Conventions

6381 The lexical conventions for regular expressions are as described in this section.

6382 Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

6383 The following tokens are processed (in addition to those string constants shown in the
6384 grammar):

6385	COLL_ELEM_SINGLE	Any single-character collating element, unless it is a META_CHAR .
6386	COLL_ELEM_MULTI	Any multi-character collating element.
6387	BACKREF	Applicable only to basic regular expressions. The character string consisting of a <backslash> character followed by a single-digit numeral, '1' to '9'.
6388		
6389		
6390	DUP_COUNT	Represents a numeric constant. It shall be an integer in the range 0 ≤ DUP_COUNT ≤{ RE_DUP_MAX }. This token is only recognized when the context of the grammar requires it. At all other times, digits not preceded by a <backslash> character are treated as ORD_CHAR .
6391		
6392		
6393		
6394	META_CHAR	One of the characters:
6395		^ When found first in a bracket expression
6396		– When found anywhere but first (after an initial '^', if any) or last in a bracket expression, or as the ending range point in a range expression
6397		
6398		
6399] When found anywhere but first (after an initial '^', if any) in a bracket expression
6400		
6401	L_ANCHOR	Applicable only to basic regular expressions. The character '^' when it appears as the first character of a basic regular expression and when not QUOTED_CHAR . The '^' may be recognized as an anchor elsewhere; see Section 9.3.8 (on page 188).
6402		
6403		
6404		
6405	ORD_CHAR	A character, other than one of the special characters in SPEC_CHAR .
6406	QUOTED_CHAR	In a BRE, one of the character sequences:
6407		\^ \. * \[\\$ \\
6408		In an ERE, one of the character sequences:
6409		\^ \. \[\\$ \(\) \
6410		* + ? { \\
6411	R_ANCHOR	(Applicable only to basic regular expressions.) The character '\$' when it appears as the last character of a basic regular expression and when not QUOTED_CHAR . The '\$' may be recognized as an anchor elsewhere; see Section 9.3.8 (on page 188).
6412		
6413		
6414		

6415 **SPEC_CHAR** For basic regular expressions, one of the following special characters:

6416 . Anywhere outside bracket expressions

6417 \ Anywhere outside bracket expressions

6418 [Anywhere outside bracket expressions

6419 ^ When used as an anchor (see [Section 9.3.8](#), on page 188)

6420 \$ When used as an anchor

6421 * Anywhere except first in an entire RE, anywhere in a

6422 bracket expression, directly following "\(", directly

6423 following an anchoring '^'

6424 For extended regular expressions, shall be one of the following

6425 special characters found anywhere outside bracket expressions:

6426 ^ . [\$ () |

6427 * + ? { \

6428 (The close-parenthesis shall be considered special in this context only

6429 if matched with a preceding open-parenthesis.)

6430 **9.5.2 RE and Bracket Expression Grammar**

6431 This section presents the grammar for basic regular expressions, including the bracket

6432 expression grammar that is common to both BREs and EREs.

```
6433   %token    ORD_CHAR QUOTED_CHAR DUP_COUNT
6434   %token    BACKREF L_ANCHOR R_ANCHOR
6435   %token    Back_open_paren Back_close_paren
6436   /*        '\('                '\)'                */
6437   %token    Back_open_brace Back_close_brace
6438   /*        '\{'                '\}'                */
6439   /* The following tokens are for the Bracket Expression
6440        grammar common to both REs and EREs. */
6441   %token    COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6442   %token    Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6443   /*        '['                ']'                '['                ']'                ':'                ':'                */
6444   %token    class_name
6445   /* class_name is a keyword to the LC_CTYPE locale category */
6446   /* (representing a character class) in the current locale */
6447   /* and is only recognized between [: and :] */
6448   %start    basic_reg_exp
6449   %%
6450   /* -----
6451        Basic Regular Expression
6452        -----
6453   */
6454   basic_reg_exp :                RE_expression
6455                | L_ANCHOR
```



```

6456         |                               R_ANCHOR
6457         | L_ANCHOR                       R_ANCHOR
6458         | L_ANCHOR RE_expression
6459         |           RE_expression R_ANCHOR
6460         | L_ANCHOR RE_expression R_ANCHOR
6461         ;
6462 RE_expression : simple_RE
6463         | RE_expression simple_RE
6464         ;
6465 simple_RE : nondupl_RE
6466         | nondupl_RE RE_dupl_symbol
6467         ;
6468 nondupl_RE : one_char_or_coll_elem_RE
6469         | Back_open_paren RE_expression Back_close_paren
6470         | BACKREF
6471         ;
6472 one_char_or_coll_elem_RE : ORD_CHAR
6473         | QUOTED_CHAR
6474         | '.'
6475         | bracket_expression
6476         ;
6477 RE_dupl_symbol : '*'
6478         | Back_open_brace DUP_COUNT           Back_close_brace
6479         | Back_open_brace DUP_COUNT ','       Back_close_brace
6480         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6481         ;
6482 /* -----
6483    Bracket Expression
6484    -----
6485 */
6486 bracket_expression : '[' matching_list ']'
6487         | '[' nonmatching_list ']'
6488         ;
6489 matching_list : bracket_list
6490         ;
6491 nonmatching_list : '^' bracket_list
6492         ;
6493 bracket_list : follow_list
6494         | follow_list '-'
6495         ;
6496 follow_list : expression_term
6497         | follow_list expression_term
6498         ;
6499 expression_term : single_expression
6500         | range_expression
6501         ;
6502 single_expression : end_range
6503         | character_class
6504         | equivalence_class
6505         ;
6506 range_expression : start_range end_range
6507         | start_range '-'

```

```

6508         ;
6509     start_range      : end_range '-'
6510         ;
6511     end_range        : COLL_ELEM_SINGLE
6512         | collating_symbol
6513         ;
6514     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6515         | Open_dot COLL_ELEM_MULTI Dot_close
6516         | Open_dot META_CHAR Dot_close
6517         ;
6518     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6519         | Open_equal COLL_ELEM_MULTI Equal_close
6520         ;
6521     character_class  : Open_colon class_name Colon_close
6522         ;

```

6523 The BRE grammar does not permit **L_ANCHOR** or **R_ANCHOR** inside "`\(`" and "`\)`" (which
6524 implies that '`^`' and '`$`' are ordinary characters). This reflects the semantic limits on the
6525 application, as noted in [Section 9.3.8](#) (on page 188). Implementations are permitted to extend the
6526 language to interpret '`^`' and '`$`' as anchors in these locations, and as such, conforming
6527 applications cannot use unescaped '`^`' and '`$`' in positions inside "`\(`" and "`\)`" that might
6528 be interpreted as anchors.

6529 9.5.3 ERE Grammar

6530 This section presents the grammar for extended regular expressions, excluding the bracket
6531 expression grammar.

6532 **Note:** The bracket expression grammar and the associated `%token` lines are identical between BREs
6533 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6534 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6535 %start  extended_reg_exp
6536 %%
6537 /* -----
6538    Extended Regular Expression
6539    -----
6540 */
6541 extended_reg_exp  :          ERE_branch
6542                 | extended_reg_exp '|' ERE_branch
6543                 ;
6544 ERE_branch       :          ERE_expression
6545                 | ERE_branch ERE_expression
6546                 ;
6547 ERE_expression   : one_char_or_coll_elem_ERE
6548                 | '^'
6549                 | '$'
6550                 | '(' extended_reg_exp ')'
6551                 | ERE_expression ERE_dupl_symbol
6552                 ;
6553 one_char_or_coll_elem_ERE : ORD_CHAR
6554                 | QUOTED_CHAR
6555                 | '.'

```

```

6556             | bracket_expression
6557             ;
6558 ERE_dupl_symbol : '*'
6559                | '+'
6560                | '?'
6561                | '{' DUP_COUNT '}'
6562                | '{' DUP_COUNT ',' '}'
6563                | '{' DUP_COUNT ',' DUP_COUNT '}'
6564             ;

```

6565 The ERE grammar does not permit several constructs that previous sections specify as having
6566 undefined results. Additionally, there are some constructs which the grammar permits but
6567 which still give undefined results:

6568 **ORD_CHAR** preceded by an unescaped <backslash> character

6569 One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following '|',
6570 '^', '(', or '\$'

6571 '{' not part of a valid *ERE_dupl_symbol*

6572 '|' appearing first or last in an ERE, or immediately following '|' or '(', or
6573 immediately preceding ')'

6574 Implementations are permitted to extend the language to allow these. Strictly Conforming
6575 applications cannot use such constructs.

Directory Structure and Devices

10.1 Directory Structure and Files

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

/ The root directory.

/dev Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

/tmp A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

/dev/null An empty data source and infinite data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

/dev/tty In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

/dev/console The **/dev/console** file is a generic name given to the system console (see [Section 3.392](#), on page 97). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of [Chapter 11](#) (on page 199).

6602 10.2 Output Devices and Terminal Types

6603 The utilities in the Shell and Utilities volume of POSIX.1-2017 historically have been
 6604 implemented on a wide range of terminal types, but a conforming implementation need not
 6605 support all features of all utilities on every conceivable terminal. POSIX.1-2017 states which
 6606 features are optional for certain classes of terminals in the individual utility description sections.
 6607 The implementation shall document in the system documentation which terminal types it
 6608 supports and which of these features and utilities are not supported by each terminal.

6609 When a feature or utility is not supported on a specific terminal type, as allowed by
 6610 POSIX.1-2017, and the implementation considers such a condition to be an error preventing use
 6611 of the feature or utility, the implementation shall indicate such conditions through diagnostic
 6612 messages or exit status values or both (as appropriate to the specific utility description) that
 6613 inform the user that the terminal type lacks the appropriate capability.

6614 POSIX.1-2017 uses a notational convention based on historical practice that identifies some of
 6615 the control characters defined in [Section 7.3.1](#) (on page 139) in a manner easily remembered by
 6616 users on many terminals. The correspondence between this “<control>-char” notation and the
 6617 actual control characters is shown in the following table. When POSIX.1-2017 refers to a
 6618 character by its <control>-name, it is referring to the actual control character shown in the Value
 6619 column of the table, which is not necessarily the exact control key sequence on all terminals.
 6620 Some terminals have keyboards that do not allow the direct transmission of all the non-
 6621 alphanumeric characters shown. In such cases, the system documentation shall describe which
 6622 data sequences transmitted by the terminal are interpreted by the system as representing the
 6623 special characters.

6624 **Table 10-1** Control Character Names

Name	Value	Name	Value
<control>-A	<SOH>	<control>-Q	<DC1>
<control>-B	<STX>	<control>-R	<DC2>
<control>-C	<ETX>	<control>-S	<DC3>
<control>-D	<EOT>	<control>-T	<DC4>
<control>-E	<ENQ>	<control>-U	<NAK>
<control>-F	<ACK>	<control>-V	<SYN>
<control>-G	<BEL>	<control>-W	<ETB>
<control>-H	<BS>	<control>-X	<CAN>
<control>-I	<HT>	<control>-Y	
<control>-J	<LF>	<control>-Z	<SUB>
<control>-K	<VT>	<control>-[<ESC>
<control>-L	<FF>	<control>-\ <control>-]	<FS>
<control>-M	<CR>	<control>-^	<GS>
<control>-N	<SO>	<control>-_	<RS>
<control>-O	<SI>	<control>-?	<US>
<control>-P	<DLE>		

6642 **Note:** The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that
 6643 the keystrokes represent control-shift-letter sequences.

6644

6645

General Terminal Interface

6646
6647
6648
6649

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

11.1 Interface Characteristics

11.1.1 Opening a Terminal Device File

6651
6652
6653
6654

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application’s standard input, output, and error files.

6655 Cases where applications do open a terminal device are as follows:

6656
6657
6658
6659
6660
6661
6662
6663
6664
6665

1. Opening `/dev/tty`, or the pathname returned by `ctermid()`, in order to obtain a file descriptor for the controlling terminal; see [Section 11.1.3](#) (on page 200).
2. Opening the slave side of a pseudo-terminal; see XSH [ptsname\(\)](#).
3. Opening a modem or similar piece of equipment connected by a serial line. In this case, the terminal parameters (see [Section 11.2](#), on page 205) may be initialized to default settings by the implementation in between the last close of the device by any process and the next open of the device, or they may persist from one use to the next. The terminal parameters can be set to values that ensure the terminal behaves in a conforming manner by means of the `O_TTY_INIT` open flag when opening a terminal device that is not already open in any process, or by executing the `stty` utility with the operand `sane`.

6666
6667
6668
6669
6670

As described in `open()`, opening a terminal device file with the `O_NONBLOCK` flag clear shall cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the terminal, or the `O_NONBLOCK` flag is specified in the `open()`, the `open()` function shall return a file descriptor without waiting for a connection to be established.

6671 11.1.2 Process Groups

6672 A terminal may have a foreground process group associated with it. This foreground process
6673 group plays a special role in handling signal-generating input characters, as discussed in [Section](#)
6674 [11.1.9](#) (on page 204).

6675 A command interpreter process supporting job control can allocate the terminal to different jobs,
6676 or process groups, by placing related processes in a single process group and associating this
6677 process group with the terminal. A terminal's foreground process group may be set or examined
6678 by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*.
6679 The terminal interface aids in this allocation by restricting access to the terminal by processes
6680 that are not in the current process group; see [Section 11.1.4](#) (on page 201).

6681 When there is no longer any process whose process ID or process group ID matches the
6682 foreground process group ID, the terminal shall have no foreground process group. It is
6683 unspecified whether the terminal has a foreground process group when there is a process whose
6684 process ID matches the foreground process group ID, but whose process group ID does not. No
6685 actions defined in POSIX.1-2017, other than allocation of a controlling terminal or a successful
6686 call to *tcsetpgrp()*, shall cause a process group to become the foreground process group of the
6687 terminal.

6688 11.1.3 The Controlling Terminal

6689 A terminal may belong to a process as its controlling terminal. Each process of a session that has
6690 a controlling terminal has the same controlling terminal. A terminal may be the controlling
6691 terminal for at most one session. The controlling terminal for a session is allocated by the session
6692 leader in an implementation-defined manner. If a session leader has no controlling terminal, and
6693 opens a terminal device file that is not already associated with a session without using the
6694 *O_NOCTTY* option (see *open()*), it is implementation-defined whether the terminal becomes the
6695 controlling terminal of the session leader. If a process which is not a session leader opens a
6696 terminal file, or the *O_NOCTTY* option is used on *open()*, then that terminal shall not become
6697 the controlling terminal of the calling process. When a controlling terminal becomes associated
6698 with a session, its foreground process group shall be set to the process group of the session
6699 leader.

6700 The controlling terminal is inherited by a child process during a *fork()* function call. A process
6701 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;
6702 other processes remaining in the old session that had this terminal as their controlling terminal
6703 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in
6704 the current session) associated with the controlling terminal, it is unspecified whether all
6705 processes that had that terminal as their controlling terminal cease to have any controlling
6706 terminal. Whether and how a session leader can reacquire a controlling terminal after the
6707 controlling terminal has been relinquished in this fashion is unspecified. A process does not
6708 relinquish its controlling terminal simply by closing all of its file descriptors associated with the
6709 controlling terminal if other processes continue to have it open.

6710 When a controlling process terminates, the controlling terminal is dissociated from the current
6711 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by
6712 other processes in the earlier session may be denied, with attempts to access the terminal treated
6713 as if a modem disconnect had been sensed.

6714 11.1.4 Terminal Access Control

6715 If a process is in the foreground process group of its controlling terminal, read operations shall
6716 be allowed, as described in [Section 11.1.5](#). Any attempts by a process in a background process
6717 group to read from its controlling terminal cause its process group to be sent a SIGTTIN signal
6718 unless one of the following special cases applies: if the reading process is ignoring the SIGTTIN
6719 signal or the reading thread is blocking the SIGTTIN signal, or if the process group of the
6720 reading process is orphaned, the `read()` shall return `-1`, with `errno` set to `[EIO]` and no signal shall
6721 be sent. The default action of the SIGTTIN signal shall be to stop the process to which it is sent.
6722 See [<signal.h>](#).

6723 If a process is in the foreground process group of its controlling terminal, write operations shall
6724 be allowed as described in [Section 11.1.8](#) (on page 203). Attempts by a process in a background
6725 process group to write to its controlling terminal shall cause the process group to be sent a
6726 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if
6727 TOSTOP is set and the process is ignoring the SIGTTOU signal or the writing thread is blocking
6728 the SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is
6729 not sent. If TOSTOP is set, the process group of the writing process is orphaned, the writing
6730 process is not ignoring the SIGTTOU signal, and the writing thread is not blocking the SIGTTOU
6731 signal, the `write()` shall return `-1`, with `errno` set to `[EIO]` and no signal shall be sent.

6732 Certain calls that set terminal parameters are treated in the same fashion as `write()`, except that
6733 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set
6734 (see [Section 11.2.5](#) (on page 210), `tcdrain()`, `tcflow()`, `tcflush()`, `tcsendbreak()`, `tcsetattr()`, and
6735 `tcsetpgrp()`).

6736 11.1.5 Input Processing and Reading Data

6737 A terminal device associated with a terminal device file may operate in full-duplex mode, so
6738 that data may arrive even while output is occurring. Each terminal device file has an input
6739 queue associated with it, into which incoming data is stored by the system before being read by
6740 a process. The system may impose a limit, `{MAX_INPUT}`, on the number of bytes that may be
6741 stored in the input queue. The behavior of the system when this limit is exceeded is
6742 implementation-defined.

6743 Two general kinds of input processing are available, determined by whether the terminal device
6744 file is in canonical mode or non-canonical mode. These modes are described in [Section 11.1.6](#) (on
6745 page 202) and [Section 11.1.7](#) (on page 202). Additionally, input characters are processed
6746 according to the `c_iflag` (see [Section 11.2.2](#), on page 206) and `c_lflag` (see [Section 11.2.5](#), on page
6747 210) fields. Such processing can include “echoing”, which in general means transmitting input
6748 characters immediately back to the terminal when they are received from the terminal. This is
6749 useful for terminals that can operate in full-duplex mode.

6750 The manner in which data is provided to a process reading from a terminal device file is
6751 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether
6752 or not the `O_NONBLOCK` flag is set by `open()` or `fcntl()`.

6753 If the `O_NONBLOCK` flag is clear, then the read request shall be blocked until data is available
6754 or a signal has been received. If the `O_NONBLOCK` flag is set, then the read request shall be
6755 completed, without blocking, in one of three ways:

- 6756 1. If there is enough data available to satisfy the entire request, the `read()` shall complete
6757 successfully and shall return the number of bytes read.

6758 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete
6759 successfully, having read as much data as possible, and shall return the number of bytes it
6760 was able to read.

6761 3. If there is no data available, the *read()* shall return -1 , with *errno* set to [EAGAIN].

6762 When data is available depends on whether the input processing mode is canonical or non-
6763 canonical. [Section 11.1.6](#) and [Section 11.1.7](#) describe each of these input processing modes.

6764 11.1.6 Canonical Mode Input Processing

6765 In canonical mode input processing, terminal input is processed in units of lines. A line is
6766 delimited by a <newline> character (NL), an end-of-file character (EOF), or an end-of-line (EOL)
6767 character. See [Section 11.1.9](#) (on page 204) for more information on EOF and EOL. This means
6768 that a read request shall not return until an entire line has been typed or a signal has been
6769 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall
6770 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even
6771 one, may be requested in a *read()* without losing information.

6772 If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes
6773 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If
6774 {MAX_CANON} is not defined, there shall be no such limit; see *pathconf()*.

6775 Erase and kill processing occur when either of two special characters, the ERASE and KILL
6776 characters (see [Section 11.1.9](#), on page 204), is received. This processing shall affect data in the
6777 input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited
6778 data makes up the current line. The ERASE character shall delete the last character in the current
6779 line, if there is one. The KILL character shall delete all data in the current line, if there is any.
6780 The ERASE and KILL characters shall have no effect if there is no data in the current line. The
6781 ERASE and KILL characters themselves shall not be placed in the input queue.

6782 11.1.7 Non-Canonical Mode Input Processing

6783 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and
6784 kill processing shall not occur. The values of the MIN and TIME members of the *c_cc* array are
6785 used to determine how to process the bytes received. POSIX.1-2017 does not specify whether
6786 the setting of O_NONBLOCK takes precedence over MIN or TIME settings. Therefore, if
6787 O_NONBLOCK is set, *read()* may return immediately, regardless of the setting of MIN or TIME.
6788 Also, if no data is available, *read()* may either return 0, or return -1 with *errno* set to [EAGAIN].

6789 MIN represents the minimum number of bytes that should be received when the *read()* function
6790 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and
6791 short-term data transmissions. If MIN is greater than {MAX_INPUT}, the response to the request
6792 is undefined. The four possible values for MIN and TIME and their interactions are described
6793 below.

6874 Case A: MIN>0, TIME>0

6875 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is
6876 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction
6877 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall
6878 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer
6879 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN
6800 bytes are received, the characters received to that point shall be returned to the user. Note that if
6801 TIME expires at least one byte shall be returned because the timer would not have been enabled
6802 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and
6803 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is
6804 in the buffer at the time of the *read()*, the result shall be as if data has been received immediately
6805 after the *read()*.

6806 Case B: MIN>0, TIME=0

6807 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A
6808 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall
6809 block until MIN bytes are received), or a signal is received. A program that uses case B to read
6810 record-based terminal I/O may block indefinitely in the read operation.

6811 Case C: MIN=0, TIME>0

6812 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read
6813 timer that shall be activated as soon as the *read()* function is processed. A read shall be satisfied
6814 as soon as a single byte is received or the read timer expires. Note that in case C if the timer
6815 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be
6816 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely
6817 waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the
6818 *read()* shall return a value of zero, having read no data. If data is in the buffer at the time of the
6819 *read()*, the timer shall be started as if data has been received immediately after the *read()*.

6820 Case D: MIN=0, TIME=0

6821 The minimum of either the number of bytes requested or the number of bytes currently
6822 available shall be returned without waiting for more bytes to be input. If no characters are
6823 available, *read()* shall return a value of zero, having read no data.

6824 11.1.8 Writing Data and Output Processing

6825 When a process writes one or more bytes to a terminal device file, they are processed according
6826 to the *c_oflag* field (see [Section 11.2.3](#), on page 207). The implementation may provide a
6827 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have
6828 been scheduled for transmission to the device, but the transmission has not necessarily
6829 completed. See *write()* for the effects of O_NONBLOCK on *write()*.

6830 **11.1.9 Special Characters**

6831 Certain characters have special functions on input or output or both. These functions are
6832 summarized as follows:

6833 **INTR** Special character on input, which is recognized if the ISIG flag is set. Generates a
6834 SIGINT signal which is sent to all processes in the foreground process group for which
6835 the terminal is the controlling terminal. If ISIG is set, the INTR character shall be
6836 discarded when processed.

6837 **QUIT** Special character on input, which is recognized if the ISIG flag is set. Generates a
6838 SIGQUIT signal which is sent to all processes in the foreground process group for
6839 which the terminal is the controlling terminal. If ISIG is set, the QUIT character shall be
6840 discarded when processed.

6841 **ERASE** Special character on input, which is recognized if the ICANON flag is set. Erases the
6842 last character in the current line; see [Section 11.1.6](#) (on page 202). It shall not erase
6843 beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is
6844 set, the ERASE character shall be discarded when processed.

6845 **KILL** Special character on input, which is recognized if the ICANON flag is set. Deletes the
6846 entire line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the KILL
6847 character shall be discarded when processed.

6848 **EOF** Special character on input, which is recognized if the ICANON flag is set. When
6849 received, all the bytes waiting to be read are immediately passed to the process without
6850 waiting for a <newline>, and the EOF is discarded. Thus, if there are no bytes waiting
6851 (that is, the EOF occurred at the beginning of a line), a byte count of zero shall be
6852 returned from the *read()*, representing an end-of-file indication. If ICANON is set, the
6853 EOF character shall be discarded when processed.

6854 **NL** Special character on input, which is recognized if the ICANON flag is set. It is the line
6855 delimiter <newline>. It cannot be changed.

6856 **EOL** Special character on input, which is recognized if the ICANON flag is set. It is an
6857 additional line delimiter, like NL.

6858 **SUSP** If the ISIG flag is set, receipt of the SUSP character shall cause a SIGTSTP signal to be
6859 sent to all processes in the foreground process group for which the terminal is the
6860 controlling terminal, and the SUSP character shall be discarded when processed.

6861 **STOP** Special character on both input and output, which is recognized if the IXON (output
6862 control) or IXOFF (input control) flag is set. Can be used to suspend output
6863 temporarily. It is useful with CRT terminals to prevent output from disappearing before
6864 it can be read. If IXON is set, the STOP character shall be discarded when processed.

6865 **START** Special character on both input and output, which is recognized if the IXON (output
6866 control) or IXOFF (input control) flag is set. Can be used to resume output that has been
6867 suspended by a STOP character. If IXON is set, the START character shall be discarded
6868 when processed.

6869 **CR** Special character on input, which is recognized if the ICANON flag is set; it is the
6870 <carriage-return> character. When ICANON and ICRNL are set and IGNCR is not set,
6871 this character shall be translated into an NL, and shall have the same effect as an NL
6872 character. It cannot be changed.

6873 The NL and CR characters cannot be changed. It is implementation-defined whether the START
6874 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and
6875 SUSP shall be changeable to suit individual tastes. Special character functions associated with

6876 changeable special control characters can be disabled individually.

6877 If two or more special characters have the same value, the function performed when that
6878 character is received is undefined.

6879 A special character is recognized not only by its value, but also by its context; for example, an
6880 implementation may support multi-byte sequences that have a meaning different from the
6881 meaning of the bytes when considered individually. Implementations may also support
6882 additional single-byte functions. These implementation-defined multi-byte or single-byte
6883 functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without
6884 interpretation, except as required to recognize the special characters defined in this section.

6885 XSI If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding
6886 `<backslash>` character, in which case no special function shall occur.

6887 11.1.10 Modem Disconnect

6888 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if
6889 CLOCAL is not set in the `c_cflag` field for the terminal (see Section 11.2.4, on page 209), the
6890 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling
6891 terminal. Unless other arrangements have been made, this shall cause the controlling process to
6892 terminate (see `exit()`). Any subsequent read from the terminal device shall return the value of
6893 zero, indicating end-of-file; see `read()`. Thus, processes that read a terminal file and test for end-
6894 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in `read()`
6895 also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent
6896 `write()` to the terminal device shall return `-1`, with `errno` set to [EIO], until the device is closed.

6897 11.1.11 Closing a Terminal Device File

6898 The last process to close a terminal device file shall cause any output to be sent to the device and
6899 shall cause any input to be discarded. If HUPCL is set in the control modes and the
6900 communications port supports a disconnect function, the terminal device shall perform a
6901 disconnect.

6902 11.2 Parameters that Can be Set

6903 11.2.1 The `termios` Structure

6904 Routines that need to control certain terminal I/O characteristics shall do so by using the
6905 `termios` structure as defined in the `<termios.h>` header.

6906 Since the `termios` structure may include additional members, and the standard members may
6907 include both standard and non-standard modes, the structure should never be initialized
6908 directly by the application as this may cause the terminal to behave in a non-conforming
6909 manner. When opening a terminal device (other than a pseudo-terminal) that is not already open
6910 in any process, it should be opened with the `O_TTY_INIT` flag before initializing the structure
6911 using `tcgetattr()` to ensure that any non-standard elements of the `termios` structure are set to
6912 values that result in conforming behavior of the terminal interface.

6913 The members of the **termios** structure include (but are not limited to):

Member Type	Array Size	Member Name	Description
6914 tcflag_t		<i>c_iflag</i>	Input modes.
6915 tcflag_t		<i>c_oflag</i>	Output modes.
6916 tcflag_t		<i>c_cflag</i>	Control modes.
6917 tcflag_t		<i>c_lflag</i>	Local modes.
6918 tcflag_t			
6919 cc_t	NCCS	<i>c_cc[]</i>	Control characters.
6920			

6921 The **tcflag_t** and **cc_t** types are defined in the **<termios.h>** header. They shall be unsigned
6922 integer types.

6923 11.2.2 Input Modes

6924 Values of the *c_iflag* field describe the basic terminal input control, and are composed of the
6925 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
6926 symbols in this table are defined in **<termios.h>**:

Mask Name	Description
6927 BRKINT	Signal interrupt on break.
6928 ICRNL	Map CR to NL on input.
6929 IGNBRK	Ignore break condition.
6930 IGNCR	Ignore CR.
6931 IGNPAR	Ignore characters with parity errors.
6932 INLCR	Map NL to CR on input.
6933 INPCK	Enable input parity check.
6934 ISTRIP	Strip character.
6935 IXANY	Enable any character to restart output.
6936 IXOFF	Enable start/stop input control.
6937 IXON	Enable start/stop output control.
6938 PARMRK	Mark parity errors.
6939	

6940 In the context of asynchronous serial data transmission, a break condition shall be defined as a
6941 sequence of zero-valued bits that continues for more than the time to send one byte. The entire
6942 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a
6943 time equivalent to more than one byte. In contexts other than asynchronous serial data
6944 transmission, the definition of a break condition is implementation-defined.

6945 If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the
6946 input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the
6947 break condition shall flush the input and output queues, and if the terminal is the controlling
6948 terminal of a foreground process group, the break condition shall generate a single SIGINT
6949 signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break
6950 condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

6951 If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

6952 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than
6953 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is
6954 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid
6955 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff
6956 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be
6957 given to the application as a single byte 0x00.

6958 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking
6959 shall be disabled, allowing output parity generation without input parity errors. Note that
6960 whether input parity checking is enabled or disabled is independent of whether parity detection
6961 is enabled or disabled (see [Section 11.2.4](#), on page 209). If parity detection is enabled but input
6962 parity checking is disabled, the hardware to which the terminal is connected shall recognize the
6963 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6964 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits
6965 shall be processed.

6966 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a
6967 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a
6968 received CR character shall be translated into an NL character.

6969 If IXANY is set, any input character shall restart output that has been suspended.

6970 If IXON is set, start/stop output control shall be enabled. A received STOP character shall
6971 suspend output and a received START character shall restart output. When IXON is set, START
6972 and STOP characters are not read, but merely perform flow control functions. When IXON is not
6973 set, the START and STOP characters shall be read.

6974 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP
6975 characters, which are intended to cause the terminal device to stop transmitting data, as needed
6976 to prevent the input queue from overflowing and causing implementation-defined behavior,
6977 and shall transmit START characters, which are intended to cause the terminal device to resume
6978 transmitting data, as soon as the device can continue transmitting data without risk of
6979 overflowing the input queue. The precise conditions under which STOP and START characters
6980 are transmitted are implementation-defined.

6981 The initial input control value after *open()* is implementation-defined.

6982 **11.2.3 Output Modes**

6983 The *c_oflag* field specifies the terminal interface's treatment of output, and is composed of the
6984 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
6985 symbols in the following table are defined in [<termios.h>](#):

Mask Name	Description
OPOST	Perform output processing.
XSI ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else NUL.
NLDLY	Select newline delays:
NL0	Newline character type 0.
NL1	Newline character type 1.
CRDLY	Select carriage-return delays:
CR0	Carriage-return delay type 0.
CR1	Carriage-return delay type 1.
CR2	Carriage-return delay type 2.
CR3	Carriage-return delay type 3.
TABDLY	Select horizontal-tab delays:
TAB0	Horizontal-tab delay type 0.
TAB1	Horizontal-tab delay type 1.
TAB2	Horizontal-tab delay type 2.
TAB3	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	Backspace-delay type 0.
BS1	Backspace-delay type 1.
VTDLY	Select vertical-tab delays:
VT0	Vertical-tab delay type 0.
VT1	Vertical-tab delay type 1.
FFDLY	Select form-feed delays:
FF0	Form-feed delay type 0.
FF1	Form-feed delay type 1.

7016 If OPOST is set, output data shall be post-processed as described below, so that lines of text are
7017 modified to appear appropriately on the terminal device; otherwise, characters shall be
7018 transmitted without change.

7019 XSI If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is
7020 set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character
7021 shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is
7022 assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays
7023 specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed
7024 function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the
7025 CR character is actually transmitted.

7026 The delay bits specify how long transmission stops to allow for mechanical or other movement
7027 when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If
7028 OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful
7029 for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character
7030 shall be DEL; otherwise, NUL.

7031 If a <form-feed> or <vertical-tab> delay is specified, it shall last for about 2 seconds.

7032 Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be
7033 used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

7034 Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be

7035 about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall
7036 transmit two fill characters, and type 2 four fill characters.

7037 Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be
7038 about 0.10 seconds. Type 3 specifies that <tab> characters shall be expanded into <space>
7039 characters. If OFILL is set, two fill characters shall be transmitted for any delay.

7040 Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be
7041 transmitted.

7042 The actual delays depend on line speed and system load.

7043 The initial output control value after *open()* is implementation-defined.

7044 11.2.4 Control Modes

7045 The *c_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-
7046 inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in
7047 this table are defined in <termios.h>; not all values specified are required to be supported by the
7048 underlying hardware (if any). If the terminal is a pseudo-terminal, it is unspecified whether non-
7049 default values are unsupported, or are supported and emulated in software, or are handled by
7050 *tcsetattr()*, *tcgetattr()*, and the *stty* utility as if they are supported but have no effect on the
7051 behavior of the terminal interface.

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

7064 In addition, the input and output baud rates are stored in the **termios** structure. The symbols in
7065 the following table are defined in <termios.h>. Not all values specified are required to be
7066 supported by the underlying hardware (if any). For pseudo-terminals, the input and output
7067 baud rates set in the **termios** structure need not affect the speed of data transmission through the
7068 terminal interface.

7069 **Note:** The term “baud” is used historically here, but is not technically correct. This is properly “bits
7070 per second”, which may not be the same as baud. However, the term is used because of the
7071 historical usage and understanding.

7072
7073
7074
7075
7076
7077
7078
7079
7080

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

7081
7082
7083
7084

The following functions are provided for getting and setting the values of the input and output baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*. The effects on the terminal device shall not become effective and not all errors need be detected until the *tcsetattr()* function is successfully called.

7085
7086
7087
7088
7089

The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read. CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used; otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

7090

If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

7091
7092
7093

If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity shall be used.

7094
7095
7096

If HUPCL is set, the modem control lines for the port shall be lowered when the last process with the port open closes the port or the process terminates. The modem connection shall be broken.

7097
7098

If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines shall be monitored.

7099
7100
7101

Under normal circumstances, a call to the *open()* function shall wait for the modem connection to complete. However, if the O_NONBLOCK flag is set (see *open()*) or if CLOCAL has been set, the *open()* function shall return immediately without waiting for the connection.

7102
7103
7104
7105

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate need not be set on the connection between that terminal and the machine to which it is directly connected.

7106

The initial hardware control value after *open()* is implementation-defined.

7107 11.2.5 Local Modes

7108
7109
7110

The *c_iflag* field of the argument structure is used to control various functions. It is composed of the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in this table are defined in [<termios.h>](#).

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

7121 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input
7122 characters shall not be echoed.

7123 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if
7124 possible, the last character in the current line from the display. If there is no character to erase, an
7125 implementation may echo an indication that this was the case, or do nothing.

7126 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the
7127 line from the display or shall echo the <newline> character after the KILL character.

7128 If ECHONL and ICANON are set, the <newline> character shall be echoed even if ECHO is not
7129 set.

7130 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit
7131 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as
7132 described in [Section 11.1.6](#) (on page 202).

7133 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall
7134 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired
7135 between bytes. The time value represents tenths of a second. See [Section 11.1.7](#) (on page 202) for
7136 more details.

7137 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is
7138 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.
7139 If IEXTEN is not set, implementation-defined functions shall not be recognized and the
7140 corresponding input characters are processed as described for ICANON, ISIG, IXON, and
7141 IXOFF.

7142 If ISIG is set, each input character shall be checked against the special control characters INTR,
7143 QUIT, and SUSP. If an input character matches one of these control characters, the function
7144 associated with that character shall be performed. If ISIG is not set, no checking shall be done.
7145 Thus these special input functions are possible only if ISIG is set.

7146 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,
7147 QUIT, and SUSP characters shall not be done.

7148 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to
7149 write to its controlling terminal if it is not in the foreground process group for that terminal. This
7150 signal, by default, stops the members of the process group. Otherwise, the output generated by
7151 that process shall be output to the current output stream. If the writing process is ignoring the
7152 SIGTTOU signal or the writing thread is blocking the SIGTTOU signal, the process is allowed to
7153 produce output, and the SIGTTOU signal shall not be sent.

7154 The initial local control value after *open()* is implementation-defined.

7155 **11.2.6 Special Control Characters**

7156 The special control character values shall be defined by the array `c_cc`. The subscript name and
 7157 description for each element in both canonical and non-canonical modes are as follows:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

7172 The subscript values are unique, except that the VMIN and VTIME subscripts may have the
 7173 same values as the VEOF and VEOL subscripts, respectively.

7174 Implementations that do not support changing the START and STOP characters may ignore the
 7175 character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when
 7176 `tcsetattr()` is called, but shall return the value in use when `tcgetattr()` is called.

7177 The initial values of all control characters are implementation-defined.

7178 If the value of one of the changeable special control characters (see [Section 11.1.9](#), on page 204) is
 7179 `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the
 7180 disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special
 7181 meaning for the VMIN and VTIME entries of the `c_cc` array.

Utility Conventions

7184 12.1 Utility Argument Syntax

7185 This section describes the argument syntax of the standard utilities and introduces terminology
7186 used throughout POSIX.1-2017 for describing the arguments processed by the utilities.

7187 Within POSIX.1-2017, a special notation is used for describing the syntax of a utility's
7188 arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated
7189 by this example (see XCU Section 2.9.1, on page 2365):

```
7190 utility_name[-a] [-b] [-c option_argument]
7191 [-d|-e] [-f[option_argument]][operand...]
```

7192 The notation used for the SYNOPSIS sections imposes requirements on the implementors of the
7193 standard utilities and provides a simple reference for the application developer or system user.

- 7194 1. The utility in the example is named *utility_name*. It is followed by options, option-
7195 arguments, and operands. The arguments that consist of <hyphen-minus> characters
7196 and single letters or digits, such as 'a', are known as "options" (or, historically, "flags").
7197 Certain options are followed by an "option-argument", as shown with [-c
7198 *option_argument*]. The arguments following the last options and option-arguments are
7199 named "operands".
- 7200 2. Option-arguments are shown separated from their options by <blank> characters, except
7201 when the option-argument is enclosed in the '[' and ']' notation to indicate that it is
7202 optional. This reflects the situation in which an optional option-argument (if present) is
7203 included within the same argument string as the option; for a mandatory option-
7204 argument, it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page
7205 216) require that the option be a separate argument from its option-argument and that
7206 option-arguments not be optional, but there are some exceptions in POSIX.1-2017 to
7207 ensure continued operation of historical applications:
 - 7208 a. If the SYNOPSIS of a standard utility shows an option with a mandatory option-
7209 argument (as with [-c *option_argument*] in the example), a conforming application
7210 shall use separate arguments for that option and its option-argument. However, a
7211 conforming implementation shall also permit applications to specify the option
7212 and option-argument in the same argument string without intervening <blank>
7213 characters.
 - 7214 b. If the SYNOPSIS shows an optional option-argument (as with
7215 [-f[*option_argument*]] in the example), a conforming application shall place any
7216 option-argument for that option directly adjacent to the option in the same
7217 argument string, without intervening <blank> characters. If the utility receives an
7218 argument containing only the option, it shall behave as specified in its description
7219 for an omitted option-argument; it shall not treat the next argument (if any) as the
7220 option-argument for that option.

- 7221 3. Options are usually listed in alphabetical order unless this would make the utility
 7222 description more confusing. There are no implied relationships between the options
 7223 based upon the order in which they appear, unless otherwise stated in the OPTIONS
 7224 section, or unless the exception in Guideline 11 of [Section 12.2](#) (on page 216) applies. If an
 7225 option that does not have option-arguments is repeated, the results are undefined, unless
 7226 otherwise stated.
- 7227 4. Frequently, names of parameters that require substitution by actual values are shown
 7228 with embedded <underscore> characters. Alternatively, parameters are shown as follows:
 7229 `<parameter name>`
- 7230 The angle brackets are used for the symbolic grouping of a phrase representing a single
 7231 parameter and conforming applications shall not include them in data submitted to the
 7232 utility.
- 7233 5. When a utility has only a few permissible options, they are sometimes shown
 7234 individually, as in the example. Utilities with many flags generally show all of the
 7235 individual flags (that do not take option-arguments) grouped, as in:
 7236 `utility_name [-abcDxyz] [-p arg] [operand]`
- 7237 Utilities with very complex arguments may be shown as follows:
 7238 `utility_name [options] [operands]`
- 7239 6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a
 7240 numeric value:
 7241 The number is interpreted as a decimal integer.
 7242 Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric
 7243 values.
 7244 When the utility description states that it accepts negative numbers as operands or
 7245 option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are
 7246 syntactically recognized as numeric values.
 7247 When the utility description states that the number is a file size-related value (such
 7248 as a file size or offset, line number, or block count), numerals in the range 0 to the
 7249 maximum file size supported by the implementation are syntactically recognized as
 7250 numeric values (see [XCU Section 1.5](#), on page 2343). Where negative values are
 7251 permitted, any value in the range -(maximum file size) to the maximum file size is
 7252 accepted.
 7253 Ranges greater than those listed here are allowed.
- 7254 This does not mean that all numbers within the allowable range are necessarily
 7255 semantically correct. A standard utility that accepts an option-argument or operand that
 7256 is to be interpreted as a number, and for which a range of values smaller than that shown
 7257 above is permitted by the POSIX.1-2017, describes that smaller range along with the
 7258 description of the option-argument or operand. If an error is generated, the utility's
 7259 diagnostic message shall indicate that the value is out of the supported range, not that it
 7260 is syntactically incorrect.
- 7261 7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and
 7262 can be omitted. Conforming applications shall not include the '[' and ']' symbols in
 7263 data submitted to the utility.

7264 8. Arguments separated by the '|' (<vertical-line>) bar notation are mutually-exclusive.
 7265 Conforming applications shall not include the '|' symbol in data submitted to the utility.
 7266 Alternatively, mutually-exclusive options and operands may be listed with multiple
 7267 synopsis lines.

7268 For example:

```
7269 utility_name -d[-a] [-c option_argument] [operand...]  

  7270 utility_name [-a] [-b] [operand...]
```

7271 When multiple synopsis lines are given for a utility, it is an indication that the utility has
 7272 mutually-exclusive arguments. These mutually-exclusive arguments alter the
 7273 functionality of the utility so that only certain other arguments are valid in combination
 7274 with one of the mutually-exclusive arguments. Only one of the mutually-exclusive
 7275 arguments is allowed for invocation of the utility. Unless otherwise stated in an
 7276 accompanying OPTIONS section, the relationships between arguments depicted in the
 7277 SYNOPSIS sections are mandatory requirements placed on conforming applications. The
 7278 use of conflicting mutually-exclusive arguments produces undefined results, unless a
 7279 utility description specifies otherwise. When an option is shown without the '[' and
 7280 ']' brackets, it means that option is required for that version of the SYNOPSIS. However,
 7281 it is not required to be the first argument, as shown in the example above, unless
 7282 otherwise stated.

7283 9. Ellipses ("...") are used to denote that one or more occurrences of an operand are
 7284 allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero
 7285 or more options or operands can be specified. The form:

```
7286 utility_name [-g option_argument]... [operand...]
```

7287 indicates that multiple occurrences of the option and its option-argument preceding the
 7288 ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See
 7289 also Guideline 11 in [Section 12.2](#) (on page 216).)

7290 The form:

```
7291 utility_name -f option_argument [-f option_argument]... [operand...]
```

7292 indicates that the -f option is required to appear at least once and may appear multiple
 7293 times.

7294 10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities
 7295 volume of POSIX.1-2017, the indented lines following the initial line are continuation
 7296 lines. An actual use of the command would appear on a single logical line.

7297 12.2 Utility Syntax Guidelines

7298 The following guidelines are established for the naming of utilities and for the specification of
7299 options, option-arguments, and operands. The *getopt()* function in the System Interfaces
7300 volume of POSIX.1-2017 assists utilities in handling options and operands that conform to these
7301 guidelines.

7302 Operands and option-arguments can contain characters not specified in the portable character
7303 set.

7304 The guidelines are intended to provide guidance to the authors of future utilities, such as those
7305 written specific to a local system or that are components of a larger application. Some of the
7306 standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections
7307 describe the deviations.

7308 **Guideline 1:** Utility names should be between two and nine characters, inclusive.

7309 **Guideline 2:** Utility names should include lowercase letters (the **lower** character
7310 classification) and digits only from the portable character set.

7311 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**
7312 character classification) from the portable character set. The **-W** (capital-W)
7313 option shall be reserved for vendor options.

7314 Multi-digit options should not be allowed.

7315 **Guideline 4:** All options should be preceded by the '-' delimiter character.

7316 **Guideline 5:** One or more options without option-arguments, followed by at most one
7317 option that takes an option-argument, should be accepted when grouped
7318 behind one '-' delimiter.

7319 **Guideline 6:** Each option and option-argument should be a separate argument, except as
7320 noted in [Section 12.1](#) (on page 213), item (2).

7321 **Guideline 7:** Option-arguments should not be optional.

7322 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they
7323 should be presented as a single argument, using <comma> characters within
7324 that argument or <blank> characters within that argument to separate them.

7325 **Guideline 9:** All options should precede operands on the command line.

7326 **Guideline 10:** The first -- argument that is not an option-argument should be accepted as a
7327 delimiter indicating the end of options. Any following arguments should be
7328 treated as operands, even if they begin with the '-' character.

7329 **Guideline 11:** The order of different options relative to one another should not matter, unless
7330 the options are documented as mutually-exclusive and such an option is
7331 documented to override any incompatible options preceding it. If an option
7332 that has option-arguments is repeated, the option and option-argument
7333 combinations should be interpreted in the order specified on the command
7334 line.

7335 **Guideline 12:** The order of operands may matter and position-related interpretations should
7336 be determined on a utility-specific basis.

7337 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading
7338 or writing, the '-' operand should be used to mean only standard input (or
7339 standard output when it is clear from context that an output file is being
7340 specified) or a file named -.

7341 **Guideline 14:** If an argument can be identified according to Guidelines 3 through 10 as an
7342 option, or as a group of options without option-arguments behind one '-'
7343 delimiter, then it should be treated as such.

7344 The utilities in the Shell and Utilities volume of POSIX.1-2017 that claim conformance to these
7345 guidelines shall conform completely to these guidelines as if these guidelines contained the term
7346 "shall" instead of "should". On some implementations, the utilities accept usage in violation of
7347 these guidelines for backwards-compatibility as well as accepting the required form.

7348 Where a utility described in the Shell and Utilities volume of POSIX.1-2017 as conforming to
7349 these guidelines is required to accept, or not to accept, the operand '-' to mean standard input
7350 or output, this usage is explained in the OPERANDS section. Otherwise, if such a utility uses
7351 operands to represent files, it is implementation-defined whether the operand '-' stands for
7352 standard input (or standard output), or for a file named -.

7353 It is recommended that all future utilities and applications use these guidelines to enhance user
7354 portability. The fact that some historical utilities could not be changed (to avoid breaking
7355 existing applications) should not deter this future goal.

7356

7357

Headers

7358 This chapter describes the contents of headers.

7359 Headers contain function prototypes, the definition of symbolic constants, common structures,
7360 preprocessor macros, and defined types. Each function in the System Interfaces volume of
7361 POSIX.1-2017 specifies the headers that an application shall include in order to use that function.
7362 In most cases, only one header is required. These headers are present on an application
7363 development system; they need not be present on the target execution system.

7364 **Format of Entries**

7365 The entries in this chapter are based on a common format as follows. The only sections relating
7366 to conformance are the SYNOPSIS and DESCRIPTION.

7367 **NAME**

7368 This section gives the name or names of the entry and briefly states its purpose.

7369 **SYNOPSIS**

7370 This section summarizes the use of the entry being described.

7371 **DESCRIPTION**

7372 This section describes the functionality of the header.

7373 **APPLICATION USAGE**

7374 This section is informative. This section gives warnings and advice to application
7375 developers about the entry. In the event of conflict between warnings and advice and a
7376 normative part of this volume of POSIX.1-2017, the normative material is to be taken as
7377 correct.

7378 **RATIONALE**

7379 This section is informative. This section contains historical information concerning the
7380 contents of this volume of POSIX.1-2017 and why features were included or discarded
7381 by the standard developers.

7382 **FUTURE DIRECTIONS**

7383 This section is informative. This section provides comments which should be used as a
7384 guide to current thinking; there is not necessarily a commitment to adopt these future
7385 directions.

7386 **SEE ALSO**

7387 This section is informative. This section gives references to related information.

7388 **CHANGE HISTORY**

7389 This section is informative. This section shows the derivation of the entry and any
7390 significant changes that have been made to it.

7391 **NAME**

7392 aio.h — asynchronous input and output

7393 **SYNOPSIS**

7394 #include <aio.h>

7395 **DESCRIPTION**7396 The **<aio.h>** header shall define the **aio_cb** structure, which shall include at least the following
7397 members:

7398	int	aio_fildes	File descriptor.
7399	off_t	aio_offset	File offset.
7400	volatile void	*aio_buf	Location of buffer.
7401	size_t	aio_nbytes	Length of transfer.
7402	int	aio_reqprio	Request priority offset.
7403	struct sigevent	aio_sigevent	Signal number and value.
7404	int	aio_lio_opcode	Operation to be performed.

7405 The **<aio.h>** header shall define the **off_t**, **pthread_attr_t**, **size_t**, and **ssize_t** types as described
7406 in **<sys/types.h>**.7407 The **<aio.h>** header shall define the **struct timespec** structure as described in **<time.h>**.7408 The **<aio.h>** header shall define the **sigevent** structure and **sigval** union as described in
7409 **<signal.h>**.7410 The **<aio.h>** header shall define the following symbolic constants:7411 **AIO_ALLDONE** A return value indicating that none of the requested operations could be
7412 canceled since they are already complete.7413 **AIO_CANCELED** A return value indicating that all requested operations have been
7414 canceled.7415 **AIO_NOTCANCELED**
7416 A return value indicating that some of the requested operations could not
7417 be canceled since they are in progress.7418 **LIO_NOP** A *lio_listio()* element operation option indicating that no transfer is
7419 requested.7420 **LIO_NOWAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7421 is to continue execution while the *lio_listio()* operation is being
7422 performed, and no notification is given when the operation is complete.7423 **LIO_READ** A *lio_listio()* element operation option requesting a read.7424 **LIO_WAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7425 is to suspend until the *lio_listio()* operation is complete.7426 **LIO_WRITE** A *lio_listio()* element operation option requesting a write.7427 The following shall be declared as functions and may also be defined as macros. Function
7428 prototypes shall be provided.

7429	int	aio_cancel(int, struct aio_cb *);
7430	int	aio_error(const struct aio_cb *);
7431	FSC SIO	int aio_fsync(int, struct aio_cb *);
7432	int	aio_read(struct aio_cb *);
7433	ssize_t	aio_return(struct aio_cb *);
7434	int	aio_suspend(const struct aio_cb *const [], int,

```

7435             const struct timespec *);
7436 int         aio_write(struct aiocb *);
7437 int         lio_listio(int, struct aiocb *restrict const [restrict], int,
7438             struct sigevent *restrict);

```

7439 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>, <signal.h>, and <time.h>.

7441 APPLICATION USAGE

7442 None.

7443 RATIONALE

7444 None.

7445 FUTURE DIRECTIONS

7446 None.

7447 SEE ALSO

7448 [<fcntl.h>](#), [<signal.h>](#), [<sys/types.h>](#), [<time.h>](#)

7449 XSH [aio_cancel\(\)](#), [aio_error\(\)](#), [aio_fsync\(\)](#), [aio_read\(\)](#), [aio_return\(\)](#), [aio_suspend\(\)](#), [aio_write\(\)](#),
7450 [fsync\(\)](#), [lio_listio\(\)](#), [lseek\(\)](#), [read\(\)](#), [write\(\)](#)

7451 CHANGE HISTORY

7452 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7453 Issue 6

7454 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7455 The description of the constants is expanded.

7456 The **restrict** keyword is added to the prototype for *lio_listio()*.

7457 Issue 7

7458 The <aio.h> header is moved from the Asynchronous Input and Output option to the Base.

7459 This reference page is clarified with respect to macros and symbolic constants, and type and structure declarations are added.

7461 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0038 [98] is applied.

7462 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0058 [579] is applied.

7463 **NAME**

7464 arpa/inet.h ‡definitions for internet operations

7465 **SYNOPSIS**

7466 #include <arpa/inet.h>

7467 **DESCRIPTION**7468 The **<arpa/inet.h>** header shall define the **in_port_t** and **in_addr_t** types as described in
7469 **<netinet/in.h>**.7470 The **<arpa/inet.h>** header shall define the **in_addr** structure as described in **<netinet/in.h>**.7471 IP6 The **<arpa/inet.h>** header shall define the **INET_ADDRSTRLEN** and **INET6_ADDRSTRLEN**
7472 macros as described in **<netinet/in.h>**.7473 The following shall be declared as functions, or defined as macros, or both. If functions are
7474 declared, function prototypes shall be provided.7475 uint32_t htonl(uint32_t);
7476 uint16_t htons(uint16_t);
7477 uint32_t ntohl(uint32_t);
7478 uint16_t ntohs(uint16_t);7479 The **<arpa/inet.h>** header shall define the **uint32_t** and **uint16_t** types as described in
7480 **<inttypes.h>**.7481 The following shall be declared as functions and may also be defined as macros. Function
7482 prototypes shall be provided.7483 in_addr_t inet_addr(const char *);
7484 char *inet_ntoa(struct in_addr);
7485 const char *inet_ntop(int, const void *restrict, char *restrict,
7486 socklen_t);
7487 int inet_pton(int, const char *restrict, void *restrict);7488 Inclusion of the **<arpa/inet.h>** header may also make visible all symbols from **<netinet/in.h>**
7489 and **<inttypes.h>**.7490 **APPLICATION USAGE**

7491 None.

7492 **RATIONALE**

7493 None.

7494 **FUTURE DIRECTIONS**

7495 None.

7496 **SEE ALSO**7497 **<inttypes.h>**, **<netinet/in.h>**7498 XSH *htonl()*, *inet_addr()*, *inet_ntop()*7499 **CHANGE HISTORY**

7500 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7501 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.7502 **Issue 7**

7503 SD5-XBD-ERN-6 is applied.

7504 **NAME**7505 `assert.h` — verify program assertion7506 **SYNOPSIS**7507 `#include <assert.h>`7508 **DESCRIPTION**

7509 CX The functionality described on this reference page is aligned with the ISO C standard. Any
7510 conflict between the requirements described here and the ISO C standard is unintentional. This
7511 volume of POSIX.1-2017 defers to the ISO C standard.

7512 The `<assert.h>` header shall define the `assert()` macro. It refers to the macro `NDEBUG` which is
7513 not defined in the header. If `NDEBUG` is defined as a macro name before the inclusion of this
7514 header, the `assert()` macro shall be defined simply as:

7515

```
#define assert(ignore)((void) 0)
```

7516 Otherwise, the macro behaves as described in `assert()`.

7517 The `assert()` macro shall be redefined according to the current state of `NDEBUG` each time
7518 `<assert.h>` is included.

7519 The `assert()` macro shall be implemented as a macro, not as a function. If the macro definition is
7520 suppressed in order to access an actual function, the behavior is undefined.

7521 **APPLICATION USAGE**

7522 None.

7523 **RATIONALE**

7524 None.

7525 **FUTURE DIRECTIONS**

7526 None.

7527 **SEE ALSO**7528 XSH [assert\(\)](#)7529 **CHANGE HISTORY**

7530 First released in Issue 1. Derived from Issue 1 of the SVID.

7531 **Issue 6**7532 The definition of the `assert()` macro is changed for alignment with the ISO/IEC 9899:1999
7533 standard.

7534 **NAME**

7535 complex.h ‡'complex arithmetic

7536 **SYNOPSIS**

7537 #include <complex.h>

7538 **DESCRIPTION**

7539 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7540 conflict between the requirements described here and the ISO C standard is unintentional. This
 7541 volume of POSIX.1-2017 defers to the ISO C standard.

7542 The **<complex.h>** header shall define the following macros:7543 complex Expands to **_Complex**.7544 _Complex_I Expands to a constant expression of type **const float _Complex**, with the value
 7545 of the imaginary unit (that is, a number i such that $i^2=-1$).7546 imaginary Expands to **_Imaginary**.7547 _Imaginary_I Expands to a constant expression of type **const float _Imaginary** with the
 7548 value of the imaginary unit.7549 I Expands to either **_Imaginary_I** or **_Complex_I**. If **_Imaginary_I** is not defined,
 7550 I expands to **_Complex_I**.7551 The macros **imaginary** and **_Imaginary_I** shall be defined if and only if the implementation
 7552 supports imaginary types.7553 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.7554 The following shall be declared as functions and may also be defined as macros. Function
 7555 prototypes shall be provided.

```

7556       double           cabs(double complex);
7557       float            cabsf(float complex);
7558       long double      cabsl(long double complex);
7559       double complex   cacos(double complex);
7560       float complex    cacosf(float complex);
7561       double complex   cacosh(double complex);
7562       float complex    cacoshf(float complex);
7563       long double complex cacoshl(long double complex);
7564       long double complex cacosl(long double complex);
7565       double           carg(double complex);
7566       float            cargf(float complex);
7567       long double      cargl(long double complex);
7568       double complex   casin(double complex);
7569       float complex    casinf(float complex);
7570       double complex   casinh(double complex);
7571       float complex    casinhf(float complex);
7572       long double complex casinhl(long double complex);
7573       long double complex casinl(long double complex);
7574       double complex   catan(double complex);
7575       float complex    catanf(float complex);
7576       double complex   catanh(double complex);
7577       float complex    catanhf(float complex);
7578       long double complex catanhl(long double complex);
7579       long double complex catanl(long double complex);

```

```
7580     double complex      ccos(double complex);
7581     float complex       ccosf(float complex);
7582     double complex      ccosh(double complex);
7583     float complex       ccoshf(float complex);
7584     long double complex ccoshl(long double complex);
7585     long double complex ccosl(long double complex);
7586     double complex      cexp(double complex);
7587     float complex       cexpf(float complex);
7588     long double complex cexpl(long double complex);
7589     double              cimag(double complex);
7590     float               cimagf(float complex);
7591     long double         cimagl(long double complex);
7592     double complex      clog(double complex);
7593     float complex       clogf(float complex);
7594     long double complex clogl(long double complex);
7595     double complex      conj(double complex);
7596     float complex       conjf(float complex);
7597     long double complex conjl(long double complex);
7598     double complex      cpow(double complex, double complex);
7599     float complex       cpowf(float complex, float complex);
7600     long double complex cpowl(long double complex, long double complex);
7601     double complex      cproj(double complex);
7602     float complex       cprojf(float complex);
7603     long double complex cprojl(long double complex);
7604     double              creal(double complex);
7605     float               crealf(float complex);
7606     long double         creall(long double complex);
7607     double complex      csin(double complex);
7608     float complex       csinf(float complex);
7609     double complex      csinh(double complex);
7610     float complex       csinhf(float complex);
7611     long double complex csinhl(long double complex);
7612     long double complex csinl(long double complex);
7613     double complex      csqrt(double complex);
7614     float complex       csqrtf(float complex);
7615     long double complex csqrtl(long double complex);
7616     double complex      ctan(double complex);
7617     float complex       ctanf(float complex);
7618     double complex      ctanh(double complex);
7619     float complex       ctanhf(float complex);
7620     long double complex ctanhl(long double complex);
7621     long double complex ctanl(long double complex);
```


7622 **APPLICATION USAGE**

7623 Values are interpreted as radians, not degrees.

7624 **RATIONALE**7625 The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the
7626 identifier *i* for other purposes. The application can use a different identifier, say *j*, for the
7627 imaginary unit by following the inclusion of the **<complex.h>** header with:7628

```
#undef I
7629 #define j _Imaginary_I
```

7630 An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a
7631 sufficiently convenient and more generally useful notation for imaginary terms. The
7632 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or
7633 notational convenience will not result in widening types.7634 On systems with imaginary types, the application has the ability to control whether use of the
7635 macro **I** introduces an imaginary type, by explicitly defining **I** to be **_Imaginary_I** or **_Complex_I**.
7636 Disallowing imaginary types is useful for some applications intended to run on
7637 implementations without support for such types.7638 The macro **_Imaginary_I** provides a test for whether imaginary types are supported.7639 The *cis()* function ($\cos(x) + I\sin(x)$) was considered but rejected because its implementation is
7640 easy and straightforward, even though some implementations could compute sine and cosine
7641 more efficiently in tandem.7642 **FUTURE DIRECTIONS**7643 The following function names and the same names suffixed with *f* or *l* are reserved for future
7644 use, and may be added to the declarations in the **<complex.h>** header.7645

```
cerf()      cexpm1()   clog2()
7646 cerfc()    clog10()  clgamma()
7647 cexp2()    clog1p()  ctgamma()
```

7648 **SEE ALSO**7649 XSH *cabs()*, *cacos()*, *cacosh()*, *carg()*, *casin()*, *casinh()*, *catan()*, *catanh()*, *ccos()*, *ccosh()*, *cexp()*,
7650 *cimag()*, *clog()*, *conj()*, *cpow()*, *cproj()*, *creal()*, *csin()*, *csinh()*, *csqrt()*, *ctan()*, *ctanh()*7651 **CHANGE HISTORY**

7652 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7653 **NAME**

7654 cpio.h — cpio archive values

7655 **SYNOPSIS**

7656 #include <cpio.h>

7657 **DESCRIPTION**

7658 The <cpio.h> header shall define the symbolic constants needed by the *c_mode* field of the *cpio*
7659 archive format, with the names and values given in the following table:

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

7681 The <cpio.h> header shall define the following symbolic constant as a string:

7682 MAGIC "070707"

7683 **APPLICATION USAGE**

7684 None.

7685 **RATIONALE**

7686 None.

7687 **FUTURE DIRECTIONS**

7688 None.

7689 **SEE ALSO**

7690 XCU *pax*

7691 **CHANGE HISTORY**

7692 First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988
7693 standard.

7694 **Issue 6**

7695 The SEE ALSO is updated to refer to *pax*.

7696 **Issue 7**

7697 The <pio.h> header is moved from the XSI option to the Base.

7698 This reference page is clarified with respect to macros and symbolic constants.

7699 **NAME**

7700 ctype.h ¶character types

7701 **SYNOPSIS**

7702 #include <ctype.h>

7703 **DESCRIPTION**

7704 CX Some of the functionality described on this reference page extends the ISO C standard.
 7705 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 7706 enable the visibility of these symbols in this header.

7707 The <ctype.h> header shall define the **locale_t** type as described in <locale.h>, representing a
 7708 locale object.

7709 The following shall be declared as functions and may also be defined as macros. Function
 7710 prototypes shall be provided for use with ISO C standard compilers.

- 7711 int isalnum(int);
- 7712 CX int isalnum_l(int, locale_t);
- 7713 int isalpha(int);
- 7714 CX int isalpha_l(int, locale_t);
- 7715 OB XSI int isascii(int);
- 7716 int isblank(int);
- 7717 CX int isblank_l(int, locale_t);
- 7718 int iscntrl(int);
- 7719 CX int iscntrl_l(int, locale_t);
- 7720 int isdigit(int);
- 7721 CX int isdigit_l(int, locale_t);
- 7722 int isgraph(int);
- 7723 CX int isgraph_l(int, locale_t);
- 7724 int islower(int);
- 7725 CX int islower_l(int, locale_t);
- 7726 int isprint(int);
- 7727 CX int isprint_l(int, locale_t);
- 7728 int ispunct(int);
- 7729 CX int ispunct_l(int, locale_t);
- 7730 int isspace(int);
- 7731 CX int isspace_l(int, locale_t);
- 7732 int isupper(int);
- 7733 CX int isupper_l(int, locale_t);
- 7734 int isxdigit(int);
- 7735 CX int isxdigit_l(int, locale_t);
- 7736 OB XSI int toascii(int);
- 7737 int tolower(int);
- 7738 CX int tolower_l(int, locale_t);
- 7739 int toupper(int);
- 7740 CX int toupper_l(int, locale_t);

7741 The <ctype.h> header shall define the following as macros:

- 7742 OB XSI int _toupper(int);
- 7743 int _tolower(int);

7744 **APPLICATION USAGE**

7745 None.

7746 **RATIONALE**

7747 None.

7748 **FUTURE DIRECTIONS**

7749 None.

7750 **SEE ALSO**7751 [<locale.h>](#)

7752 XSH Section 2.2 (on page 472), [isalnum\(\)](#), [isalpha\(\)](#), [isascii\(\)](#), [isblank\(\)](#), [iscntrl\(\)](#), [isdigit\(\)](#),
7753 [isgraph\(\)](#), [islower\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#), [isupper\(\)](#), [isxdigit\(\)](#), [mblen\(\)](#), [mbstowcs\(\)](#),
7754 [mbtowc\(\)](#), [setlocale\(\)](#), [toascii\(\)](#), [tolower\(\)](#), [_tolower\(\)](#), [toupper\(\)](#), [_toupper\(\)](#), [wcstombs\(\)](#), [wctomb\(\)](#)

7755 **CHANGE HISTORY**

7756 First released in Issue 1. Derived from Issue 1 of the SVID.

7757 **Issue 6**

7758 Extensions beyond the ISO C standard are marked.

7759 **Issue 7**7760 SD5-XBD-ERN-6 is applied, updating the wording regarding the function declarations for
7761 consistency.7762 The *_I() functions are added from The Open Group Technical Standard, 2006, Extended API Set
7763 Part 4.

7764 **NAME**

7765 dirent.h — format of directory entries

7766 **SYNOPSIS**

7767 #include <dirent.h>

7768 **DESCRIPTION**

7769 The internal format of directories is unspecified.

7770 The <dirent.h> header shall define the following type:

7771 **DIR** A type representing a directory stream. The **DIR** type may be an incomplete type.

7772 It shall also define the structure **dirent** which shall include the following members:

7773 XSI	ino_t	d_ino	File serial number.
7774	char	d_name[]	Filename string of entry.

7775 XSI The <dirent.h> header shall define the **ino_t** type as described in <sys/types.h>.

7776 The array *d_name* is of unspecified size, but shall contain a filename of at most {NAME_MAX} bytes followed by a terminating null byte.

7778 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```

7780 int          alphasort(const struct dirent **, const struct dirent **);
7781 int          closedir(DIR *);
7782 int          dirfd(DIR *);
7783 DIR          *fdopendir(int);
7784 DIR          *opendir(const char *);
7785 struct dirent *readdir(DIR *);
7786 int          readdir_r(DIR *restrict, struct dirent *restrict,
7787                 struct dirent **restrict);
7788 void         rewinddir(DIR *);
7789 int          scandir(const char *, struct dirent ***,
7790                    int (*)(const struct dirent *),
7791                    int (*)(const struct dirent **,
7792                             const struct dirent **));
7793 XSI void     seekdir(DIR *, long);
7794 long         telldir(DIR *);
    
```

7795 **APPLICATION USAGE**

7796 None.

7797 **RATIONALE**

7798 Information similar to that in the <dirent.h> header is contained in a file <sys/dir.h> in 4.2 BSD and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of POSIX.1-2017 is **struct direct**. The filename was changed because the name <sys/dir.h> was also used in earlier implementations to refer to definitions related to the older access method; this produced name conflicts. The name of the structure was changed because this volume of POSIX.1-2017 does not completely define what is in the structure, so it could be different on some implementations from **struct direct**.

7805 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:

7806 sizeof(d_name)

7807 is incorrect; use:

7808 `strlen(d_name)`

7809 instead.

7810 The array of **char** *d_name* is not a fixed size. Implementations may need to declare **struct dirent**
7811 with an array size for *d_name* of 1, but the actual number of bytes provided matches (or only
7812 slightly exceeds) the length of the filename string.

7813 **FUTURE DIRECTIONS**

7814 None.

7815 **SEE ALSO**

7816 [<sys/types.h>](#)

7817 XSH *alphasort()*, *closedir()*, *dirfd()*, *fdopendir()*, *readdir()*, *rewinddir()*, *seekdir()*, *telldir()*

7818 **CHANGE HISTORY**

7819 First released in Issue 2.

7820 **Issue 5**

7821 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7822 **Issue 6**

7823 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.

7824 The **restrict** keyword is added to the prototype for *readdir_r()*.

7825 **Issue 7**

7826 The *alphasort()*, *dirfd()*, and *scandir()* functions are added from The Open Group Technical
7827 Standard, 2006, Extended API Set Part 1.

7828 The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API
7829 Set Part 2.

7830 Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying the definition of the **DIR**
7831 type.

7832 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0039 [291], XBD/TC1-2008/0040 [291],
7833 XBD/TC1-2008/0041 [291], and XBD/TC1-2008/0042 [206] are applied.

7834 **NAME**

7835 dlfcn.h ‡dynamic linking

7836 **SYNOPSIS**

7837 #include <dlfcn.h>

7838 **DESCRIPTION**7839 The <dlfcn.h> header shall define at least the following symbolic constants for use in the
7840 construction of a *dlopen()* *mode* argument:

7841	RTLD_LAZY	Relocations are performed at an implementation-defined time.
7842	RTLD_NOW	Relocations are performed when the object is loaded.
7843	RTLD_GLOBAL	All symbols are available for relocation processing of other modules.
7844	RTLD_LOCAL	All symbols are not made available for relocation processing by other
7845		modules.

7846 The following shall be declared as functions and may also be defined as macros. Function
7847 prototypes shall be provided.

```
7848           int     dlclose(void *);
7849           char    *dLError(void);
7850           void    *dlopen(const char *, int);
7851           void    *dlsym(void *restrict, const char *restrict);
```

7852 **APPLICATION USAGE**

7853 None.

7854 **RATIONALE**

7855 None.

7856 **FUTURE DIRECTIONS**

7857 None.

7858 **SEE ALSO**7859 XSH *dlclose()*, *dLError()*, *dlopen()*, *dlsym()*7860 **CHANGE HISTORY**

7861 First released in Issue 5.

7862 **Issue 6**7863 The **restrict** keyword is added to the prototype for *dlsym()*.7864 **Issue 7**

7865 The <dlfcn.h> header is moved from the XSI option to the Base.

7866 This reference page is clarified with respect to macros and symbolic constants.

7867 **NAME**7868 `errno.h` — system error numbers7869 **SYNOPSIS**7870 `#include <errno.h>`7871 **DESCRIPTION**

7872 CX Some of the functionality described on this reference page extends the ISO C standard. Any
7873 conflict between the requirements described here and the ISO C standard is unintentional. This
7874 volume of POSIX.1-2017 defers to the ISO C standard.

7875 The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7876 The **<errno.h>** header shall provide a declaration or definition for *errno*. The symbol *errno* shall
7877 expand to a modifiable lvalue of type **int**. It is unspecified whether *errno* is a macro or an
7878 identifier declared with external linkage. If a macro definition is suppressed in order to access an
7879 actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

7880 The **<errno.h>** header shall define the following macros which shall expand to integer constant
7881 expressions with type **int**, distinct positive values (except as noted below), and which shall be
7882 suitable for use in **#if** preprocessing directives:

7883	[E2BIG]	Argument list too long.
7884	[EACCES]	Permission denied.
7885	[EADDRINUSE]	Address in use.
7886	[EADDRNOTAVAIL]	Address not available.
7887	[EAFNOSUPPORT]	Address family not supported.
7888	[EAGAIN]	Resource unavailable, try again (may be the same value as 7889 [EWOULDBLOCK]).
7890	[EALREADY]	Connection already in progress.
7891	[EBADF]	Bad file descriptor.
7892	[EBADMSG]	Bad message.
7893	[EBUSY]	Device or resource busy.
7894	[ECANCELED]	Operation canceled.
7895	[ECHILD]	No child processes.
7896	[ECONNABORTED]	Connection aborted.
7897	[ECONNREFUSED]	Connection refused.
7898	[ECONNRESET]	Connection reset.
7899	[EDEADLK]	Resource deadlock would occur.
7900	[EDESTADDRREQ]	Destination address required.
7901	[EDOM]	Mathematics argument out of domain of function.
7902	[EDQUOT]	Reserved.
7903	[EEXIST]	File exists.

7904		[EFAULT]	Bad address.
7905		[EFBIG]	File too large.
7906		[EHOSTUNREACH]	Host is unreachable.
7907		[EIDRM]	Identifier removed.
7908		[EILSEQ]	Illegal byte sequence.
7909		[EINPROGRESS]	Operation in progress.
7910		[EINTR]	Interrupted function.
7911		[EINVAL]	Invalid argument.
7912		[EIO]	I/O error.
7913		[EISCONN]	Socket is connected.
7914		[EISDIR]	Is a directory.
7915		[ELOOP]	Too many levels of symbolic links.
7916		[EMFILE]	File descriptor value too large.
7917		[EMLINK]	Too many links.
7918		[EMSGSIZE]	Message too large.
7919		[EMULTIHOP]	Reserved.
7920		[ENAMETOOLONG]	Filename too long.
7921		[ENETDOWN]	Network is down.
7922		[ENETRESET]	Connection aborted by network.
7923		[ENETUNREACH]	Network unreachable.
7924		[ENFILE]	Too many files open in system.
7925		[ENOBUFS]	No buffer space available.
7926	OB XSR	[ENODATA]	No message is available on the STREAM head read queue.
7927		[ENODEV]	No such device.
7928		[ENOENT]	No such file or directory.
7929		[ENOEXEC]	Executable file format error.
7930		[ENOLCK]	No locks available.
7931		[ENOLINK]	Reserved.
7932		[ENOMEM]	Not enough space.
7933		[ENOMSG]	No message of the desired type.
7934		[ENOPROTOOPT]	Protocol not available.
7935		[ENOSPC]	No space left on device.
7936	OB XSR	[ENOSR]	No STREAM resources.

7937	OB XSR	[ENOSTR]	Not a STREAM.
7938		[ENOSYS]	Functionality not supported.
7939		[ENOTCONN]	The socket is not connected.
7940		[ENOTDIR]	Not a directory or a symbolic link to a directory.
7941		[ENOTEMPTY]	Directory not empty.
7942		[ENOTRECOVERABLE]	
7943			State not recoverable.
7944		[ENOTSOCK]	Not a socket.
7945		[ENOTSUP]	Not supported (may be the same value as [EOPNOTSUPP]).
7946		[ENOTTY]	Inappropriate I/O control operation.
7947		[ENXIO]	No such device or address.
7948		[EOPNOTSUPP]	Operation not supported on socket (may be the same value as
7949			[ENOTSUP]).
7950		[EOVERFLOW]	Value too large to be stored in data type.
7951		[EOWNERDEAD]	Previous owner died.
7952		[EPERM]	Operation not permitted.
7953		[EPIPE]	Broken pipe.
7954		[EPROTO]	Protocol error.
7955		[EPROTONOSUPPORT]	
7956			Protocol not supported.
7957		[EPROTOTYPE]	Protocol wrong type for socket.
7958		[ERANGE]	Result too large.
7959		[EROFS]	Read-only file system.
7960		[ESPIPE]	Invalid seek.
7961		[ESRCH]	No such process.
7962		[ESTALE]	Reserved.
7963	OB XSR	[ETIME]	Stream <i>ioctl()</i> timeout.
7964		[ETIMEDOUT]	Connection timed out.
7965		[ETXTBSY]	Text file busy.
7966		[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7967		[EXDEV]	Cross-device link.

7968 APPLICATION USAGE

7969 Additional error numbers may be defined on conforming systems; see the System Interfaces
7970 volume of POSIX.1-2017.

7971 RATIONALE

7972 None.

7973 FUTURE DIRECTIONS

7974 None.

7975 SEE ALSO

7976 XSH [Section 2.3](#) (on page 481)

7977 CHANGE HISTORY

7978 First released in Issue 1. Derived from Issue 1 of the SVID.

7979 Issue 5

7980 Updated for alignment with the POSIX Realtime Extension.

7981 Issue 6

7982 The following new requirements on POSIX implementations derive from alignment with the
7983 Single UNIX Specification:

7984 The majority of the error conditions previously marked as extensions are now mandatory,
7985 except for the STREAMS-related error conditions.

7986 Values for *errno* are now required to be distinct positive values rather than non-zero values. This
7987 change is for alignment with the ISO/IEC 9899: 1999 standard.

7988 Issue 7

7989 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and
7990 [EOPNOTSUPP] to be the same values.

7991 The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group
7992 Technical Standard, 2006, Extended API Set Part 2.

7993 Functionality relating to the XSI STREAMS option is marked obsolescent.

7994 Functionality relating to the Threads option is moved to the Base.

7995 This reference page is clarified with respect to macros and symbolic constants.

7996 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0043 [324] is applied.

7997 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0059 [496] is applied.

7998 **NAME**7999 `fcntl.h` — file control options8000 **SYNOPSIS**8001 `#include <fcntl.h>`8002 **DESCRIPTION**

8003 The **<fcntl.h>** header shall define the following symbolic constants for the *cmd* argument used
8004 by *fcntl()*. The values shall be unique and shall be suitable for use in **#if** preprocessing
8005 directives.

8006 `F_DUPFD` Duplicate file descriptor.

8007 `F_DUPFD_CLOEXEC`

8008 Duplicate file descriptor with the close-on-exec flag `FD_CLOEXEC` set.

8009 `F_GETFD` Get file descriptor flags.

8010 `F_SETFD` Set file descriptor flags.

8011 `F_GETFL` Get file status flags and file access modes.

8012 `F_SETFL` Set file status flags.

8013 `F_GETLK` Get record locking information.

8014 `F_SETLK` Set record locking information.

8015 `F_SETLKW` Set record locking information; wait if blocked.

8016 `F_GETOWN` Get process or process group ID to receive SIGURG signals.

8017 `F_SETOWN` Set process or process group ID to receive SIGURG signals.

8018 The **<fcntl.h>** header shall define the following symbolic constant used for the *fcntl()* file
8019 descriptor flags, which shall be suitable for use in **#if** preprocessing directives.

8020 `FD_CLOEXEC` Close the file descriptor upon execution of an *exec* family function.

8021 The **<fcntl.h>** header shall also define the following symbolic constants for the *l_type* argument
8022 used for record locking with *fcntl()*. The values shall be unique and shall be suitable for use in
8023 **#if** preprocessing directives.

8024 `F_RDLCK` Shared or read lock.

8025 `F_UNLCK` Unlock.

8026 `F_WRLCK` Exclusive or write lock.

8027 The **<fcntl.h>** header shall define the values used for *l_whence*, `SEEK_SET`, `SEEK_CUR`, and
8028 `SEEK_END` as described in **<stdio.h>**.

8029 The **<fcntl.h>** header shall define the following symbolic constants as file creation flags for use
8030 in the *oflag* value to *open()* and *openat()*. The values shall be bitwise-distinct and shall be
8031 suitable for use in **#if** preprocessing directives.

8032 `O_CLOEXEC` The `FD_CLOEXEC` flag associated with the new descriptor shall be set to close
8033 the file descriptor upon execution of an *exec* family function.

8034 `O_CREAT` Create file if it does not exist.

8035 `O_DIRECTORY` Fail if file is a non-directory file.

8036 O_EXCL Exclusive use flag.

8037 O_NOCTTY Do not assign controlling terminal.

8038 O_NOFOLLOW Do not follow symbolic links.

8039 O_TRUNC Truncate flag.

8040 O_TTY_INIT Set the **termios** structure terminal parameters to a state that provides
8041 conforming behavior; see [Section 11.2](#) (on page 205).

8042 The O_TTY_INIT flag can have the value zero and in this case it need not be bitwise-distinct
8043 from the other flags.

8044 The <fcntl.h> header shall define the following symbolic constants for use as file status flags for
8045 *open()*, *openat()*, and *fcntl()*. The values shall be suitable for use in **#if** preprocessing directives.

8046 O_APPEND Set append mode.

8047 SIO O_DSYNC Write according to synchronized I/O data integrity completion.

8048 O_NONBLOCK Non-blocking mode.

8049 SIO O_RSYNC Synchronized read I/O operations.

8050 O_SYNC Write according to synchronized I/O file integrity completion.

8051 The <fcntl.h> header shall define the following symbolic constant for use as the mask for file
8052 access modes. The value shall be suitable for use in **#if** preprocessing directives.

8053 O_ACCMODE Mask for file access modes.

8054 The <fcntl.h> header shall define the following symbolic constants for use as the file access
8055 modes for *open()*, *openat()*, and *fcntl()*. The values shall be unique, except that O_EXEC and
8056 O_SEARCH may have equal values. The values shall be suitable for use in **#if** preprocessing
8057 directives.

8058 O_EXEC Open for execute only (non-directory files). The result is unspecified if this
8059 flag is applied to a directory.

8060 O_RDONLY Open for reading only.

8061 O_RDWR Open for reading and writing.

8062 O_SEARCH Open directory for search only. The result is unspecified if this flag is applied
8063 to a non-directory file.

8064 O_WRONLY Open for writing only.

8065 The <fcntl.h> header shall define the symbolic constants for file modes for use as values of
8066 **mode_t** as described in <sys/stat.h>.

8067 The <fcntl.h> header shall define the following symbolic constant as a special value used in
8068 place of a file descriptor for the **at()* functions which take a directory file descriptor as a
8069 parameter:

8070 AT_FDCWD Use the current working directory to determine the target of relative file paths.

8071 The <fcntl.h> header shall define the following symbolic constant as a value for the *flag* used by
8072 *faccessat()*:

8073 AT_EACCESS Check access using effective user and group ID.

8074 The <fcntl.h> header shall define the following symbolic constant as a value for the *flag* used by

8075 *fstatat()*, *fchmodat()*, *fchownat()*, and *utimensat()*:

8076 AT_SYMLINK_NOFOLLOW

8077 Do not follow symbolic links.

8078 The **<fcntl.h>** header shall define the following symbolic constant as a value for the flag used by *linkat()*:

8080 AT_SYMLINK_FOLLOW

8081 Follow symbolic link.

8082 The **<fcntl.h>** header shall define the following symbolic constant as a value for the flag used by *unlinkat()*:

8084 AT_REMOVEDIR

8085 Remove directory instead of file.

8086 ADV The **<fcntl.h>** header shall define the following symbolic constants for the *advice* argument used by *posix_fadvise()*:

8088 POSIX_FADV_DONTNEED

8089 The application expects that it will not access the specified data in the near future.

8090 POSIX_FADV_NOREUSE

8091 The application expects to access the specified data once and then not reuse it thereafter.

8092 POSIX_FADV_NORMAL

8093 The application has no advice to give on its behavior with respect to the specified data. It is the default characteristic if no advice is given for an open file.

8095 POSIX_FADV_RANDOM

8096 The application expects to access the specified data in a random order.

8097 POSIX_FADV_SEQUENTIAL

8098 The application expects to access the specified data sequentially from lower offsets to higher offsets.

8100 POSIX_FADV_WILLNEED

8101 The application expects to access the specified data in the near future.

8102 The **<fcntl.h>** header shall define the **flock** structure describing a file lock. It shall include the following members:

8104 short l_type Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.

8105 short l_whence Flag for starting offset.

8106 off_t l_start Relative offset in bytes.

8107 off_t l_len Size; if 0 then until EOF.

8108 pid_t l_pid Process ID of the process holding the lock; returned with F_GETLK.

8109 The **<fcntl.h>** header shall define the **mode_t**, **off_t**, and **pid_t** types as described in **<sys/types.h>**.

8111 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

8113 int creat(const char *, mode_t);

8114 int fcntl(int, int, ...);

8115 int open(const char *, int, ...);

8116 int openat(int, const char *, int, ...);

```
8117 ADV      int  posix_fadvise(int, off_t, off_t, int);
8118          int  posix_fallocate(int, off_t, off_t);
```

8119 Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
8120 <unistd.h>.

8121 APPLICATION USAGE

8122 Although no existing implementation defines AT_SYMLINK_FOLLOW and
8123 AT_SYMLINK_NOFOLLOW as the same numeric value, POSIX.1-2017 does not prohibit that as
8124 the two constants are not used with the same interfaces.

8125 RATIONALE

8126 While many of the symbolic constants introduced in the <fcntl.h> header do not strictly need to
8127 be used in #if preprocessor directives, widespread historic practice has defined them as macros
8128 that are usable in such constructs, and examination of existing applications has shown that they
8129 are occasionally used in such a way. Therefore it was decided to retain this requirement on an
8130 implementation in POSIX.1-2017.

8131 FUTURE DIRECTIONS

8132 None.

8133 SEE ALSO

8134 <stdio.h>, <sys/stat.h>, <sys/types.h>, <unistd.h>

8135 XSH *creat()*, *exec*, *fcntl()*, *futimens()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*

8136 CHANGE HISTORY

8137 First released in Issue 1. Derived from Issue 1 of the SVID.

8138 Issue 5

8139 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

8140 Issue 6

8141 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

8142 O_DSYNC and O_RSYNC are marked as part of the Synchronized Input and Output
8143 option.

8144 The following new requirements on POSIX implementations derive from alignment with the
8145 Single UNIX Specification:

8146 The definition of the **mode_t**, **off_t**, and **pid_t** types is mandated.

8147 The F_GETOWN and F_SETOWN values are added for sockets.

8148 The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with
8149 IEEE Std 1003.1d-1999.

8150 IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix_madvise()* to
8151 <sys/mman.h>.

8152 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for
8153 *posix_fadvise()* and *posix_fallocate()* to be large file-aware, using **off_t** instead of **size_t**.

8154 Issue 7

8155 Austin Group Interpretation 1003.1-2001 #144 is applied, adding the O_TTY_INIT flag.

8156 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the
8157 FD_CLOEXEC flag atomically at *open()*, and adding the F_DUPFD_CLOEXEC flag.

8158 The *openat()* function is added from The Open Group Technical Standard, 2006, Extended API
8159 Set Part 2.

8160 Additional flags are added to support *faccessat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*,
8161 *open()*, *openat()*, and *unlinkat()*.

8162 This reference page is clarified with respect to macros and symbolic constants.

8163 Changes are made related to support for finegrained timestamps.

8164 Changes are made to allow a directory to be opened for searching.

8165 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0044 [274] and XBD/TC1-2008/0045
8166 [78,432] are applied.

8167 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0060 [847] is applied.

8168 **NAME**
 8169 fenv.h — floating-point environment

8170 **SYNOPSIS**
 8171 #include <fenv.h>

8172 **DESCRIPTION**

8173 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 8174 conflict between the requirements described here and the ISO C standard is unintentional. This
 8175 volume of POSIX.1-2017 defers to the ISO C standard.

8176 The <fenv.h> header shall define the following data types through **typedef**:

8177 **fenv_t** Represents the entire floating-point environment. The floating-point environment
 8178 refers collectively to any floating-point status flags and control modes supported
 8179 by the implementation.

8180 **fexcept_t** Represents the floating-point status flags collectively, including any status the
 8181 implementation associates with the flags. A floating-point status flag is a system
 8182 variable whose value is set (but never cleared) when a floating-point exception is
 8183 raised, which occurs as a side-effect of exceptional floating-point arithmetic to
 8184 provide auxiliary information. A floating-point control mode is a system variable
 8185 whose value may be set by the user to affect the subsequent behavior of floating-
 8186 point arithmetic.

8187 The <fenv.h> header shall define each of the following macros if and only if the implementation
 8188 supports the floating-point exception by means of the floating-point functions *feclearexcept()*,
 8189 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. The defined macros shall
 8190 expand to integer constant expressions with values that are bitwise-distinct.

8191 FE_DIVBYZERO
 8192 FE_INEXACT
 8193 FE_INVALID
 8194 FE_OVERFLOW
 8195 FE_UNDERFLOW

8196 MX If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be
 8197 defined. Additional implementation-defined floating-point exceptions with macros beginning
 8198 with FE_ and an uppercase letter may also be specified by the implementation.

8199 The <fenv.h> header shall define the macro FE_ALL_EXCEPT as the bitwise-inclusive OR of all
 8200 floating-point exception macros defined by the implementation, if any. If no such macros are
 8201 defined, then the macro FE_ALL_EXCEPT shall be defined as zero.

8202 The <fenv.h> header shall define each of the following macros if and only if the implementation
 8203 supports getting and setting the represented rounding direction by means of the *fegetround()*
 8204 and *fesetround()* functions. The defined macros shall expand to integer constant expressions
 8205 whose values are distinct non-negative values.

8206 FE_DOWNWARD
 8207 FE_TONEAREST
 8208 FE_TOWARDZERO
 8209 FE_UPWARD

8210 MX If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be
 8211 defined. Additional implementation-defined rounding directions with macros beginning with
 8212 FE_ and an uppercase letter may also be specified by the implementation.

8213 The **<fenv.h>** header shall define the following macro, which represents the default floating-
 8214 point environment (that is, the one installed at program startup) and has type pointer to const-
 8215 qualified **fenv_t**. It can be used as an argument to the functions within the **<fenv.h>** header that
 8216 manage the floating-point environment.

8217 `FE_DFL_ENV`

8218 The following shall be declared as functions and may also be defined as macros. Function
 8219 prototypes shall be provided.

```
8220 int feenableexcept(int);
8221 int fegetenv(fenv_t *);
8222 int fegetexceptflag(fexcept_t *, int);
8223 int fegetround(void);
8224 int feholdexcept(fenv_t *);
8225 int feraiseexcept(int);
8226 int fesetenv(const fenv_t *);
8227 int fesetexceptflag(const fexcept_t *, int);
8228 int fesetround(int);
8229 int fetestexcept(int);
8230 int feupdateenv(const fenv_t *);
```

8231 The **FENV_ACCESS** pragma provides a means to inform the implementation when an
 8232 application might access the floating-point environment to test floating-point status flags or run
 8233 under non-default floating-point control modes. The pragma shall occur either outside external
 8234 declarations or preceding all explicit declarations and statements inside a compound statement.
 8235 When outside external declarations, the pragma takes effect from its occurrence until another
 8236 **FENV_ACCESS** pragma is encountered, or until the end of the translation unit. When inside a
 8237 compound statement, the pragma takes effect from its occurrence until another **FENV_ACCESS**
 8238 pragma is encountered (including within a nested compound statement), or until the end of the
 8239 compound statement; at the end of a compound statement the state for the pragma is restored to
 8240 its condition just before the compound statement. If this pragma is used in any other context, the
 8241 behavior is undefined. If part of an application tests floating-point status flags, sets floating-
 8242 point control modes, or runs under non-default mode settings, but was translated with the state
 8243 for the **FENV_ACCESS** pragma off, the behavior is undefined. The default state (on or off) for
 8244 the pragma is implementation-defined. (When execution passes from a part of the application
 8245 translated with **FENV_ACCESS** off to a part translated with **FENV_ACCESS** on, the state of the
 8246 floating-point status flags is unspecified and the floating-point control modes have their default
 8247 settings.)

8248 APPLICATION USAGE

8249 This header is designed to support the floating-point exception status flags and directed-
 8250 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-
 8251 point state information. Also it is designed to facilitate code portability among all systems.

8252 Certain application programming conventions support the intended model of use for the
 8253 floating-point environment:

8254 A function call does not alter its caller's floating-point control modes, clear its caller's
 8255 floating-point status flags, nor depend on the state of its caller's floating-point status flags
 8256 unless the function is so documented.

8257 A function call is assumed to require default floating-point control modes, unless its
 8258 documentation promises otherwise.

8259 A function call is assumed to have the potential for raising floating-point exceptions,
8260 unless its documentation promises otherwise.

8261 With these conventions, an application can safely assume default floating-point control modes
8262 (or be unaware of them). The responsibilities associated with accessing the floating-point
8263 environment fall on the application that does so explicitly.

8264 Even though the rounding direction macros may expand to constants corresponding to the
8265 values of FLT_ROUNDS, they are not required to do so.

8266 For example:

```
8267 #include <fenv.h>
8268 void f(double x)
8269 {
8270     #pragma STDC FENV_ACCESS ON
8271     void g(double);
8272     void h(double);
8273     /* ... */
8274     g(x + 1);
8275     h(x + 1);
8276     /* ... */
8277 }
```

8278 If the function `g()` might depend on status flags set as a side-effect of the first `x+1`, or if the
8279 second `x+1` might depend on control modes set as a side-effect of the call to function `g()`, then
8280 the application shall contain an appropriately placed invocation as follows:

```
8281 #pragma STDC FENV_ACCESS ON
```

8282 RATIONALE

8283 The `fexcept_t` Type

8284 `fexcept_t` does not have to be an integer type. Its values must be obtained by a call to
8285 `fegetexceptflag()`, and cannot be created by logical operations from the exception macros. An
8286 implementation might simply implement `fexcept_t` as an `int` and use the representations
8287 reflected by the exception macros, but is not required to; other representations might contain
8288 extra information about the exceptions. `fexcept_t` might be a `struct` with a member for each
8289 exception (that might hold the address of the first or last floating-point instruction that caused
8290 that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an
8291 `fexcept_t`, and so the user cannot inspect it.

8292 Exception and Rounding Macros

8293 Macros corresponding to unsupported modes and rounding directions are not defined by the
8294 implementation and must not be defined by the application. An application might use `#ifdef`
8295 to test for this.

8296 FUTURE DIRECTIONS

8297 None.

8298 SEE ALSO

8299 XSH [*feclearexcept\(\)*](#), [*fegetenv\(\)*](#), [*fegetexceptflag\(\)*](#), [*fegetround\(\)*](#), [*feholdexcept\(\)*](#), [*feraiseexcept\(\)*](#),
8300 [*fetestexcept\(\)*](#), [*feupdateenv\(\)*](#)

8301 **CHANGE HISTORY**

8302 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

8303 The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*,
8304 *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the
8305 ISO/IEC 9899:1999 standard, Defect Report 202.8306 **Issue 7**

8307 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #37 (SD5-XBD-ERN-49) is applied.

8308 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #36 is applied.

8309 SD5-XBD-ERN-48 and SD5-XBD-ERN-69 are applied.

8310 This reference page is clarified with respect to macros and symbolic constants.

8311 **NAME**

8312 float.h ‡floating types

8313 **SYNOPSIS**

8314 #include <float.h>

8315 **DESCRIPTION**

8316 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 8317 conflict between the requirements described here and the ISO C standard is unintentional. This
 8318 volume of POSIX.1-2017 defers to the ISO C standard.

8319 The characteristics of floating types are defined in terms of a model that describes a
 8320 representation of floating-point numbers and values that provide information about an
 8321 implementation’s floating-point arithmetic.

8322 The following parameters are used to define the model for each floating-point type:

8323 *s* Sign (± 1).

8324 *b* Base or radix of exponent representation (an integer > 1).

8325 *e* Exponent (an integer between a minimum e_{\min} and a maximum e_{\max}).

8326 *p* Precision (the number of base-*b* digits in the significand).

8327 f_k Non-negative integers less than *b* (the significand digits).

8328 A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

8329 In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to
 8330 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ($x \neq 0$,
 8331 $e = e_{\min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are
 8332 not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-
 8333 Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a
 8334 floating-point exception; a *signaling NaN* generally raises a floating-point exception when
 8335 occurring as an arithmetic operand.

8336 An implementation may give zero and non-numeric values, such as infinities and NaNs, a sign,
 8337 or may leave them unsigned. Wherever such values are unsigned, any requirement in
 8338 POSIX.1-2017 to retrieve the sign shall produce an unspecified sign and any requirement to set
 8339 the sign shall be ignored.

8340 The accuracy of the floating-point operations ('+', '-', '*', '/') and of the functions in
 8341 <math.h> and <complex.h> that return floating-point results is implementation-defined, as is
 8342 the accuracy of the conversion between floating-point internal representations and string
 8343 representations performed by the functions in <stdio.h>, <stdlib.h>, and <wchar.h>. The
 8344 implementation may state that the accuracy is unknown.

8345 All integer values in the <float.h> header, except FLT_ROUNDS, shall be constant expressions
 8346 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.
 8347 All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have
 8348 separate names for all three floating-point types. The floating-point model representation is
 8349 provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

8350 The rounding mode for floating-point addition is characterized by the implementation-defined

8351 value of FLT_ROUNDS:

8352 -1 Indeterminable.

8353 0 Toward zero.

8354 1 To nearest.

8355 2 Toward positive infinity.

8356 3 Toward negative infinity.

8357 All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.

8358 The values of operations with floating operands and values subject to the usual arithmetic
8359 conversions and of floating constants are evaluated to a format whose range and precision may
8360 be greater than required by the type. The use of evaluation formats is characterized by the
8361 implementation-defined value of FLT_EVAL_METHOD:

8362 -1 Indeterminable.

8363 0 Evaluate all operations and constants just to the range and precision of the type.

8364 1 Evaluate operations and constants of type **float** and **double** to the range and precision of
8365 the **double** type; evaluate **long double** operations and constants to the range and precision
8366 of the **long double** type.

8367 2 Evaluate all operations and constants to the range and precision of the **long double** type.

8368 All other negative values for FLT_EVAL_METHOD characterize implementation-defined
8369 behavior.

8370 The **<float.h>** header shall define the following values as constant expressions with
8371 implementation-defined values that are greater or equal in magnitude (absolute value) to those
8372 shown, with the same sign.

8373 Radix of exponent representation, b .

8374 FLT_RADIX 2

8375 Number of base-FLT_RADIX digits in the floating-point significand, p .

8376 FLT_MANT_DIG

8377 DBL_MANT_DIG

8378 LDBL_MANT_DIG

8379 Number of decimal digits, n , such that any floating-point number in the widest supported
8380 floating type with p_{\max} radix b digits can be rounded to a floating-point number with n
8381 decimal digits and back again without change to the value.

$$\left\lceil \begin{array}{l} p_{\max} \log_{10} b \quad \text{if } b \text{ is a power of } 10 \\ 1 + p_{\max} \log_{10} b \quad \text{otherwise} \end{array} \right\rceil$$

8382 DECIMAL_DIG 10

8383 Number of decimal digits, q , such that any floating-point number with q decimal digits can
 8384 be rounded into a floating-point number with p radix b digits and back again without
 8385 change to the q decimal digits.

$$\left\lceil \begin{array}{ll} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lfloor (p - 1) \log_{10} b \rfloor & \text{otherwise} \end{array} \right\rceil$$

8386 FLT_DIG 6

8387 DBL_DIG 10

8388 LDBL_DIG 10

8389 Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a
 8390 normalized floating-point number, e_{\min} .

8391 FLT_MIN_EXP

8392 DBL_MIN_EXP

8393 LDBL_MIN_EXP

8394 Minimum negative integer such that 10 raised to that power is in the range of normalized
 8395 floating-point numbers.

$$\left\lceil \log_{10} b^{e_{\min} - 1} \right\rceil$$

8396 FLT_MIN_10_EXP -37

8397 DBL_MIN_10_EXP -37

8398 LDBL_MIN_10_EXP -37

8399 Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable
 8400 finite floating-point number, e_{\max} .

8401 FLT_MAX_EXP

8402 DBL_MAX_EXP

8403 LDBL_MAX_EXP

8404 CX Additionally, FLT_MAX_EXP shall be at least as large as FLT_MANT_DIG,
 8405 DBL_MAX_EXP shall be at least as large as DBL_MANT_DIG, and LDBL_MAX_EXP shall
 8406 be at least as large as LDBL_MANT_DIG; which has the effect that FLT_MAX, DBL_MAX,
 8407 and LDBL_MAX are integral.

8408 Maximum integer such that 10 raised to that power is in the range of representable finite
 8409 floating-point numbers.

$$\left\lfloor \log_{10}((1 - b^{-p}) b^{e_{\max}}) \right\rfloor$$

8410 FLT_MAX_10_EXP +37

8411 DBL_MAX_10_EXP +37

8412 LDBL_MAX_10_EXP +37

8413 The **<float.h>** header shall define the following values as constant expressions with
 8414 implementation-defined values that are greater than or equal to those shown:

8415 Maximum representable finite floating-point number.

$$(1 - b^{-p}) b^{\ell_{\max}}$$

8416 FLT_MAX 1E+37

8417 DBL_MAX 1E+37

8418 LDBL_MAX 1E+37

8419 The **<float.h>** header shall define the following values as constant expressions with
 8420 implementation-defined (positive) values that are less than or equal to those shown:

8421 The difference between 1 and the least value greater than 1 that is representable in the
 8422 given floating-point type, b^{1-p} .

8423 FLT_EPSILON 1E-5

8424 DBL_EPSILON 1E-9

8425 LDBL_EPSILON 1E-9

8426 Minimum normalized positive floating-point number, $b^{\ell_{\min}-1}$.

8427 FLT_MIN 1E-37

8428 DBL_MIN 1E-37

8429 LDBL_MIN 1E-37

8430 APPLICATION USAGE

8431 None.

8432 RATIONALE

8433 All known hardware floating-point formats satisfy the property that the exponent range is larger
 8434 than the number of mantissa digits. The ISO C standard permits a floating-point format where
 8435 this property is not true, such that the largest finite value would not be integral; however, it is
 8436 unlikely that there will ever be hardware support for such a floating-point format, and it
 8437 introduces boundary cases that portable programs should not have to be concerned with (for
 8438 example, a non-integral DBL_MAX means that *ceil()* would have to worry about overflow).
 8439 Therefore, this standard imposes an additional requirement that the largest representable finite
 8440 value is integral.

8441 FUTURE DIRECTIONS

8442 None.

8443 SEE ALSO

8444 [<complex.h>](#), [<math.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<wchar.h>](#)

8445 **CHANGE HISTORY**

8446 First released in Issue 4. Derived from the ISO C standard.

8447 **Issue 6**

8448 The description of the operations with floating-point values is updated for alignment with the
8449 ISO/IEC 9899:1999 standard.

8450 **Issue 7**

8451 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #4 (SD5-XBD-ERN-50) and #5
8452 (SD5-XBD-ERN-51) are applied.

8453 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0046 [346] and XBD/TC1-2008/0047
8454 [346] are applied.

8455 **NAME**

8456 `fmtmsg.h` — message display structures

8457 **SYNOPSIS**

8458 XSI `#include <fmtmsg.h>`

8459 **DESCRIPTION**

8460 The <fmtmsg.h> header shall define the following symbolic constants:

- 8461 `MM_HARD` Source of the condition is hardware.
- 8462 `MM_SOFT` Source of the condition is software.
- 8463 `MM_FIRM` Source of the condition is firmware.
- 8464 `MM_APPL` Condition detected by application.
- 8465 `MM_UTIL` Condition detected by utility.
- 8466 `MM_OPSYS` Condition detected by operating system.
- 8467 `MM_RECOVER` Recoverable error.
- 8468 `MM_NRECOV` Non-recoverable error.
- 8469 `MM_HALT` Error causing application to halt.
- 8470 `MM_ERROR` Application has encountered a non-fatal fault.
- 8471 `MM_WARNING` Application has detected unusual non-error condition.
- 8472 `MM_INFO` Informative message.
- 8473 `MM_NOSEV` No severity level provided for the message.
- 8474 `MM_PRINT` Display message on standard error.
- 8475 `MM_CONSOLE` Display message on system console.

8476 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The
8477 <fmtmsg.h> header shall define the symbolic constants in the **Identifier** column, which shall
8478 have the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	<code>MM_NULLLBL</code>
<i>severity</i>	int	0	<code>MM_NULLSEV</code>
<i>class</i>	long	0L	<code>MM_NULLMC</code>
<i>text</i>	char *	(char*)0	<code>MM_NULLTXT</code>
<i>action</i>	char *	(char*)0	<code>MM_NULLACT</code>
<i>tag</i>	char *	(char*)0	<code>MM_NULLTAG</code>

8486 The <fmtmsg.h> header shall also define the following symbolic constants for use as return
8487 values for *fmtmsg()*:

- 8488 `MM_OK` The function succeeded.
- 8489 `MM_NOTOK` The function failed completely.
- 8490 `MM_NOMSG` The function was unable to generate a message on standard error, but
8491 otherwise succeeded.

8492 MM_NOCON The function was unable to generate a console message, but otherwise
8493 succeeded.

8494 The following shall be declared as a function and may also be defined as a macro. A function
8495 prototype shall be provided.

```
8496           int fmtmsg(long, const char *, int,  
8497           const char *, const char *, const char *);
```

8498 **APPLICATION USAGE**

8499 None.

8500 **RATIONALE**

8501 None.

8502 **FUTURE DIRECTIONS**

8503 None.

8504 **SEE ALSO**

8505 XSH *fmtmsg()*

8506 **CHANGE HISTORY**

8507 First released in Issue 4, Version 2.

8508 **Issue 7**

8509 This reference page is clarified with respect to macros and symbolic constants.

8510 **NAME**8511 `fnmatch.h` ‡filename-matching types8512 **SYNOPSIS**8513 `#include <fnmatch.h>`8514 **DESCRIPTION**8515 The **<fnmatch.h>** header shall define the following symbolic constants:8516 **FNM_NOMATCH** The string does not match the specified pattern.8517 **FNM_PATHNAME** `<slash>` in *string* only matches `<slash>` in *pattern*.8518 **FNM_PERIOD** Leading `<period>` in *string* must be exactly matched by `<period>` in
8519 *pattern*.8520 **FNM_NOESCAPE** Disable backslash escaping.8521 The following shall be declared as a function and may also be defined as a macro. A function
8522 prototype shall be provided.8523 `int fnmatch(const char *, const char *, int);`8524 **APPLICATION USAGE**

8525 None.

8526 **RATIONALE**

8527 None.

8528 **FUTURE DIRECTIONS**

8529 None.

8530 **SEE ALSO**8531 XSH *fnmatch()*8532 **CHANGE HISTORY**

8533 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8534 **Issue 6**8535 The **FNM_NOSYS** constant is marked obsolescent.8536 **Issue 7**8537 The obsolescent **FNM_NOSYS** constant is removed.

8538 This reference page is clarified with respect to macros and symbolic constants.

8539 **NAME**

8540 ftw.h — file tree traversal

8541 **SYNOPSIS**

8542 XSI `#include <ftw.h>`

8543 **DESCRIPTION**

8544 The <ftw.h> header shall define the **FTW** structure, which shall include at least the following
8545 members:

8546 `int base`
8547 `int level`

8548 The <ftw.h> header shall define the following symbolic constants for use as values of the third
8549 argument to the application-supplied function that is passed as the second argument to *ftw()*
8550 and *nftw()*:

- 8551 **FTW_F** Non-directory file.
- 8552 **FTW_D** Directory.
- 8553 **FTW_DNR** Directory without read permission.
- 8554 **FTW_DP** Directory with subdirectories visited.
- 8555 **FTW_NS** Unknown type; *stat()* failed.
- 8556 **FTW_SL** Symbolic link.
- 8557 **FTW_SLN** Symbolic link that names a nonexistent file.

8558 The <ftw.h> header shall define the following symbolic constants for use as values of the fourth
8559 argument to *nftw()*:

- 8560 **FTW_PHYS** Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows
8561 links but does not walk down any path that crosses itself.
- 8562 **FTW_MOUNT** The walk does not cross a mount point.
- 8563 **FTW_DEPTH** All subdirectories are visited before the directory itself.
- 8564 **FTW_CHDIR** The walk changes to each directory before reading it.

8565 The following shall be declared as functions and may also be defined as macros. Function
8566 prototypes shall be provided.

8567 OB `int ftw(const char *, int (*)(const char *, const struct stat *,`
8568 `int), int);`
8569 `int nftw(const char *, int (*)(const char *, const struct stat *,`
8570 `int, struct FTW *), int, int);`

8571 The <ftw.h> header shall define the **stat** structure and the symbolic names for *st_mode* and the
8572 file type test macros as described in <sys/stat.h>.

8573 Inclusion of the <ftw.h> header may also make visible all symbols from <sys/stat.h>.

8574 APPLICATION USAGE

8575 None.

8576 RATIONALE

8577 None.

8578 FUTURE DIRECTIONS

8579 None.

8580 SEE ALSO

8581 [<sys/stat.h>](#)

8582 XSH *ftw()*, *nftw()*

8583 CHANGE HISTORY

8584 First released in Issue 1. Derived from Issue 1 of the SVID.

8585 Issue 5

8586 A description of FTW_DP is added.

8587 Issue 7

8588 The *ftw()* function is marked obsolescent.

8589 This reference page is clarified with respect to macros and symbolic constants.

8590 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0048 [403] is applied.

8591 **NAME**

8592 glob.h ‡pathname pattern-matching types

8593 **SYNOPSIS**

8594 #include <glob.h>

8595 **DESCRIPTION**8596 The <glob.h> header shall define the structures and symbolic constants used by the *glob()*
8597 function.8598 The <glob.h> header shall define the **glob_t** structure type, which shall include at least the
8599 following members:8600 size_t gl_pathc Count of paths matched by *pattern*.
8601 char **gl_pathv Pointer to a list of matched pathnames.
8602 size_t gl_offs Slots to reserve at the beginning of *gl_pathv*.8603 The <glob.h> header shall define the **size_t** type as described in <sys/types.h>.8604 The <glob.h> header shall define the following symbolic constants as values for the *flags*
8605 argument:8606 GLOB_APPEND Append generated pathnames to those previously obtained.
8607 GLOB_DOOFFS Specify how many null pointers to add to the beginning of *gl_pathv*.
8608 GLOB_ERR Cause *glob()* to return on error.
8609 GLOB_MARK Each pathname that is a directory that matches *pattern* has a <slash>
8610 appended.
8611 GLOB_NOCHECK If *pattern* does not match any pathname, then return a list consisting of
8612 only *pattern*.
8613 GLOB_NOESCAPE Disable backslash escaping.
8614 GLOB_NOSORT Do not sort the pathnames returned.

8615 The <glob.h> header shall define the following symbolic constants as error return values:

8616 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (*errfunc)()
8617 returned non-zero.
8618 GLOB_NOMATCH The *pattern* does not match any existing pathname, and
8619 GLOB_NOCHECK was not set in *flags*.
8620 GLOB_NOSPACE An attempt to allocate memory failed.8621 The following shall be declared as functions and may also be defined as macros. Function
8622 prototypes shall be provided.8623 int glob(const char *restrict, int, int (*)(const char *, int),
8624 glob_t *restrict);
8625 void globfree(glob_t *);

8626 APPLICATION USAGE

8627 None.

8628 RATIONALE

8629 None.

8630 FUTURE DIRECTIONS

8631 None.

8632 SEE ALSO

8633 [<sys/types.h>](#)

8634 XSH *glob()*

8635 CHANGE HISTORY

8636 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8637 Issue 6

8638 The **restrict** keyword is added to the prototype for *glob()*.

8639 The GLOB_NOSYS constant is marked obsolescent.

8640 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the *glob()*
8641 prototype definition by removing the **restrict** qualifier from the function pointer argument.

8642 Issue 7

8643 SD5-XBD-ERN-56 is applied, adding a reference to [<sys/types.h>](#) for the **size_t**

8644 The obsolescent GLOB_NOSYS constant is removed.

8645 This reference page is clarified with respect to macros and symbolic constants.

8646 **NAME**

8647 grp.h — group structure

8648 **SYNOPSIS**

8649 #include <grp.h>

8650 **DESCRIPTION**

8651 The <grp.h> header shall declare the **group** structure, which shall include the following
8652 members:

8653 char *gr_name The name of the group.
8654 gid_t gr_gid Numerical group ID.
8655 char **gr_mem Pointer to a null-terminated array of character
8656 pointers to member names.

8657 The <grp.h> header shall define the **gid_t** and **size_t** types as described in <sys/types.h>.

8658 The following shall be declared as functions and may also be defined as macros. Function
8659 prototypes shall be provided.

```
8660 XSI void endgrent(void);
8661 struct group *getgrent(void);
8662 struct group *getgrgid(gid_t);
8663 int getgrgid_r(gid_t, struct group *, char *,
8664 size_t, struct group **);
8665 struct group *getgrnam(const char *);
8666 int getgrnam_r(const char *, struct group *, char *,
8667 size_t , struct group **);
8668 XSI void setgrent(void);
```

8669 **APPLICATION USAGE**

8670 None.

8671 **RATIONALE**

8672 None.

8673 **FUTURE DIRECTIONS**

8674 None.

8675 **SEE ALSO**

8676 <sys/types.h>

8677 XSH *endgrent()*, *getgrgid()*, *getgrnam()*

8678 **CHANGE HISTORY**

8679 First released in Issue 1.

8680 **Issue 5**

8681 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8682 **Issue 6**

8683 The following new requirements on POSIX implementations derive from alignment with the
8684 Single UNIX Specification:

8685 The definition of **gid_t** is mandated.

8686 The *getgrgid_r()* and *getgrnam_r()* functions are marked as part of the Thread-Safe
8687 Functions option.

8688 **Issue 7**

8689 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

8690 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0049 [24] is applied.

8691 **NAME**

8692 iconv.h — codeset conversion facility

8693 **SYNOPSIS**

8694 #include <iconv.h>

8695 **DESCRIPTION**

8696 The <iconv.h> header shall define the following types:

8697 **iconv_t** Identifies the conversion from one codeset to another.8698 **size_t** As described in <sys/types.h>.8699 The following shall be declared as functions and may also be defined as macros. Function
8700 prototypes shall be provided.

```
8701 size_t iconv(iconv_t, char **restrict, size_t *restrict,  
8702             char **restrict, size_t *restrict);  
8703 int iconv_close(iconv_t);  
8704 iconv_t iconv_open(const char *, const char *);
```

8705 **APPLICATION USAGE**

8706 None.

8707 **RATIONALE**

8708 None.

8709 **FUTURE DIRECTIONS**

8710 None.

8711 **SEE ALSO**

8712 <sys/types.h>

8713 XSH *iconv()*, *iconv_close()*, *iconv_open()*8714 **CHANGE HISTORY**

8715 First released in Issue 4.

8716 **Issue 6**8717 The **restrict** keyword is added to the prototype for *iconv()*.8718 **Issue 7**8719 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

8720 The <iconv.h> header is moved from the XSI option to the Base.

8721 NAME

8722 inttypes.h ‡fixed size integer types

8723 SYNOPSIS

8724 #include <inttypes.h>

8725 DESCRIPTION

8726 CX Some of the functionality described on this reference page extends the ISO C standard.
8727 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
8728 enable the visibility of these symbols in this header.

8729 The <inttypes.h> header shall include the <stdint.h> header.

8730 The <inttypes.h> header shall define at least the following types:

8731 **imaxdiv_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8732 CX **wchar_t** As described in <stddef.h>.

8733 The <inttypes.h> header shall define the following macros. Each expands to a character string
8734 literal containing a conversion specifier, possibly modified by a length modifier, suitable for use
8735 within the *format* argument of a formatted input/output function when converting the
8736 corresponding integer type. These macros have the general form of PRI (character string literals
8737 for the *fprintf()* and *fwprintf()* family of functions) or SCN (character string literals for the
8738 *fscanf()* and *fwscanf()* family of functions), followed by the conversion specifier, followed by a
8739 name corresponding to a similar type name in <stdint.h>. In these names, *N* represents the
8740 width of the type as described in <stdint.h>. For example, *PRIdFAST32* can be used in a format
8741 string to print the value of an integer of type **int_fast32_t**.

8742 The *fprintf()* macros for signed integers are:

8743	PRIdN	PRIdLEASTN	PRIdFASTN	PRIdMAX	PRIdPTR
8744	PRiN	PRiLEASTN	PRiFASTN	PRiMAX	PRiPTR

8745 The *fprintf()* macros for unsigned integers are:

8746	PRIoN	PRIoLEASTN	PRIoFASTN	PRIoMAX	PRIoPTR
8747	PRiUN	PRiULEASTN	PRiUFASTN	PRiUMAX	PRiUPTR
8748	PRIXN	PRIXLEASTN	PRIXFASTN	PRIXMAX	PRIXPTR
8749	PRiXN	PRiXLEASTN	PRiXFASTN	PRiXMAX	PRiXPTR

8750 The *fscanf()* macros for signed integers are:

8751	SCNdN	SCNdLEASTN	SCNdFASTN	SCNdMAX	SCNdPTR
8752	SCNiN	SCNiLEASTN	SCNiFASTN	SCNiMAX	SCNiPTR

8753 The *fscanf()* macros for unsigned integers are:

8754	SCNoN	SCNoLEASTN	SCNoFASTN	SCNoMAX	SCNoPTR
8755	SCNuN	SCNuLEASTN	SCNuFASTN	SCNuMAX	SCNuPTR
8756	SCNxN	SCNxLEASTN	SCNxFASTN	SCNxMAX	SCNxPTR

8757 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and
8758 *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be
8759 defined unless the implementation does not have a suitable modifier for the type.

8760 The following shall be declared as functions and may also be defined as macros. Function
8761 prototypes shall be provided.

```

8762     intmax_t  imaxabs(intmax_t);
8763     imaxdiv_t imaxdiv(intmax_t, intmax_t);
8764     intmax_t  strtoumax(const char *restrict, char **restrict, int);
8765     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8766     intmax_t  wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);
8767     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

8768 EXAMPLES

```

8769     #include <inttypes.h>
8770     #include <wchar.h>
8771     int main(void)
8772     {
8773         uintmax_t i = UINTMAX_MAX; // This type always exists.
8774         wprintf(L"The largest integer value is %020"
8775             PRIxMAX "\n", i);
8776         return 0;
8777     }

```

8778 APPLICATION USAGE

8779 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent
8780 across machines and independent of operating systems and other implementation
8781 idiosyncrasies. It defines, through **typedef**, integer types of various sizes. Implementations are
8782 free to **typedef** them as ISO C standard integer types or extensions that they support. Consistent
8783 use of this header will greatly increase the portability of applications across platforms.

8784 RATIONALE

8785 The ISO/IEC 9899:1990 standard specified that the language should support four signed and
8786 unsigned integer data types **char**, **short**, **int**, and **long** ‡but placed very little ¶requirement on
8787 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8788 not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits
8789 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to
8790 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems
8791 for users who migrate from one system to another which assigns different sizes to integer types,
8792 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.
8793 The need for defining an extended integer type increased with the introduction of 64-bit
8794 systems.

8795 FUTURE DIRECTIONS

8796 Macro names beginning with PRI or SCN followed by any lowercase letter or 'X' may be added
8797 to the macros defined in the <inttypes.h> header.

8798 SEE ALSO

8799 [<stddef.h>](#)

8800 XSH Section 2.2 (on page 472), [imaxabs\(\)](#), [imaxdiv\(\)](#), [strtoumax\(\)](#), [wcstoumax\(\)](#)

8801 CHANGE HISTORY

8802 First released in Issue 5.

8803 Issue 6

8804 The Open Group Base Resolution bwg97-006 is applied.

8805 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

8806 **Issue 7**
8807

POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0050 [211] is applied.

8808 **NAME**

8809 iso646.h ‡'alternative spellings

8810 **SYNOPSIS**

8811 #include <iso646.h>

8812 **DESCRIPTION**

8813 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 8814 conflict between the requirements described here and the ISO C standard is unintentional. This
 8815 volume of POSIX.1-2017 defers to the ISO C standard.

8816 The <iso646.h> header shall define the following eleven macros (on the left) that expand to the
 8817 corresponding tokens (on the right):

- 8818 and &&
- 8819 and_eq &=
- 8820 bitand &
- 8821 bitor |
- 8822 compl ~
- 8823 not !
- 8824 not_eq !=
- 8825 or ||
- 8826 or_eq |=
- 8827 xor ^
- 8828 xor_eq ^=

8829 **APPLICATION USAGE**

8830 None.

8831 **RATIONALE**

8832 None.

8833 **FUTURE DIRECTIONS**

8834 None.

8835 **SEE ALSO**

8836 None.

8837 **CHANGE HISTORY**

8838 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8839 **NAME**

8840 langinfo.h ‡language information constants

8841 **SYNOPSIS**

8842 #include <langinfo.h>

8843 **DESCRIPTION**8844 The **<langinfo.h>** header shall define the symbolic constants used to identify items of *langinfo*
8845 data (see *nl_langinfo()*).8846 The **<langinfo.h>** header shall define the **locale_t** type as described in **<locale.h>**.8847 The **<langinfo.h>** header shall define the **nl_item** type as described in **<nl_types.h>**.8848 The **<langinfo.h>** header shall define the following symbolic constants with type **nl_item**. The
8849 entries under **Category** indicate in which *setlocale()* category each item is defined.

8850	Constant	Category	Meaning
8851	CODESET	LC_CTYPE	Codeset name.
8852	D_T_FMT	LC_TIME	String for formatting date and time.
8853	D_FMT	LC_TIME	Date format string.
8854	T_FMT	LC_TIME	Time format string.
8855	T_FMT_AMPM	LC_TIME	a.m. or p.m. time format string.
8856	AM_STR	LC_TIME	Ante-meridiam affix.
8857	PM_STR	LC_TIME	Post-meridiam affix.
8858	DAY_1	LC_TIME	Name of the first day of the week (for example, Sunday).
8859	DAY_2	LC_TIME	Name of the second day of the week (for example, Monday).
8860	DAY_3	LC_TIME	Name of the third day of the week (for example, Tuesday).
8861	DAY_4	LC_TIME	Name of the fourth day of the week
8862			(for example, Wednesday).
8863	DAY_5	LC_TIME	Name of the fifth day of the week (for example, Thursday).
8864	DAY_6	LC_TIME	Name of the sixth day of the week (for example, Friday).
8865	DAY_7	LC_TIME	Name of the seventh day of the week
8866			(for example, Saturday).
8867	ABDAY_1	LC_TIME	Abbreviated name of the first day of the week.
8868	ABDAY_2	LC_TIME	Abbreviated name of the second day of the week.
8869	ABDAY_3	LC_TIME	Abbreviated name of the third day of the week.
8870	ABDAY_4	LC_TIME	Abbreviated name of the fourth day of the week.
8871	ABDAY_5	LC_TIME	Abbreviated name of the fifth day of the week.
8872	ABDAY_6	LC_TIME	Abbreviated name of the sixth day of the week.
8873	ABDAY_7	LC_TIME	Abbreviated name of the seventh day of the week.
8874	MON_1	LC_TIME	Name of the first month of the year.
8875	MON_2	LC_TIME	Name of the second month.
8876	MON_3	LC_TIME	Name of the third month.
8877	MON_4	LC_TIME	Name of the fourth month.
8878	MON_5	LC_TIME	Name of the fifth month.
8879	MON_6	LC_TIME	Name of the sixth month.
8880	MON_7	LC_TIME	Name of the seventh month.
8881	MON_8	LC_TIME	Name of the eighth month.
8882	MON_9	LC_TIME	Name of the ninth month.
8883	MON_10	LC_TIME	Name of the tenth month.
8884	MON_11	LC_TIME	Name of the eleventh month.
8885	MON_12	LC_TIME	Name of the twelfth month.
8886	ABMON_1	LC_TIME	Abbreviated name of the first month.

Constant	Category	Meaning
ABMON_2	LC_TIME	Abbreviated name of the second month.
ABMON_3	LC_TIME	Abbreviated name of the third month.
ABMON_4	LC_TIME	Abbreviated name of the fourth month.
ABMON_5	LC_TIME	Abbreviated name of the fifth month.
ABMON_6	LC_TIME	Abbreviated name of the sixth month.
ABMON_7	LC_TIME	Abbreviated name of the seventh month.
ABMON_8	LC_TIME	Abbreviated name of the eighth month.
ABMON_9	LC_TIME	Abbreviated name of the ninth month.
ABMON_10	LC_TIME	Abbreviated name of the tenth month.
ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
ERA	LC_TIME	Era description segments.
ERA_D_FMT	LC_TIME	Era date format string.
ERA_D_T_FMT	LC_TIME	Era date and time format string.
ERA_T_FMT	LC_TIME	Era time format string.
ALT_DIGITS	LC_TIME	Alternative symbols for digits.
RADIXCHAR	LC_NUMERIC	Radix character.
THOUSEP	LC_NUMERIC	Separator for thousands.
YESEXPR	LC_MESSAGES	Affirmative response expression.
NOEXPR	LC_MESSAGES	Negative response expression.
CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").

If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of `nl_langinfo(CRNCYSTR)` and `nl_langinfo_l(CRNCYSTR,loc)` is unspecified.

The following shall be declared as a function and may also be defined as a macro. A function prototype shall be provided.

```
char *nl_langinfo(nl_item);
char *nl_langinfo_l(nl_item, locale_t);
```

Inclusion of the <langinfo.h> header may also make visible all symbols from <nl_types.h>.

APPLICATION USAGE

Wherever possible, users are advised to use functions compatible with those in the ISO C standard to access items of *langinfo* data. In particular, the `strftime()` function should be used to access date and time information defined in category `LC_TIME`. The `localeconv()` function should be used to access information corresponding to `RADIXCHAR`, `THOUSEP`, and `CRNCYSTR`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 7 (on page 135), <locale.h>, <nl_types.h>

XSH `nl_langinfo()`, `localeconv()`, `strfmon()`, `strftime()`

8933 **CHANGE HISTORY**

8934 First released in Issue 2.

8935 **Issue 5**

8936 The constants YESSTR and NOSTR are marked LEGACY.

8937 **Issue 6**

8938 The constants YESSTR and NOSTR are removed.

8939 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to
8940 the "Meaning" column entry for the CRNCYSTR constant. This change is to accommodate
8941 historic practice.8942 **Issue 7**8943 The **<langinfo.h>** header is moved from the XSI option to the Base.8944 The `nl_langinfo_l()` function is added from The Open Group Technical Standard, 2006, Extended
8945 API Set Part 4.8946 This reference page is clarified with respect to macros and symbolic constants, and a declaration
8947 for the `locale_t` type is added.

8948 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0051 [107] is applied.

8949 **NAME**
8950 libgen.h ‡definitions for pattern matching functions

8951 **SYNOPSIS**
8952 XSI #include <libgen.h>

8953 **DESCRIPTION**
8954 The following shall be declared as functions and may also be defined as macros. Function
8955 prototypes shall be provided.

8956 char *basename(char *);
8957 char *dirname(char *);

8958 **APPLICATION USAGE**
8959 None.

8960 **RATIONALE**
8961 None.

8962 **FUTURE DIRECTIONS**
8963 None.

8964 **SEE ALSO**
8965 XSH *basename()*, *dirname()*

8966 **CHANGE HISTORY**
8967 First released in Issue 4, Version 2.

8968 **Issue 5**
8969 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first
8970 argument is of type **char** * rather than **const char** *.

8971 **Issue 6**
8972 The **__loc1** symbol and the *regcmp()* and *regex()* functions are removed.

8973 **NAME**

8974 limits.h ‡implementation-defined constants

8975 **SYNOPSIS**

8976 #include <limits.h>

8977 **DESCRIPTION**

8978 CX Some of the functionality described on this reference page extends the ISO C standard.
 8979 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 8980 enable the visibility of these symbols in this header.

8981 Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such
 8982 symbols are not shown as CX shaded, except under the heading “Numerical Limits”.

8983 The **<limits.h>** header shall define macros and symbolic constants for various limits. Different
 8984 categories of limits are described below, representing various limits on resources that the
 8985 implementation imposes on applications. All macros and symbolic constants defined in this
 8986 header shall be suitable for use in **#if** preprocessing directives.

8987 Implementations may choose any appropriate value for each limit, provided it is not more
 8988 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names
 8989 beginning with **_POSIX** may be found in **<unistd.h>**.

8990 Applications should not assume any particular value for a limit. To achieve maximum
 8991 portability, an application should not require more resource than the Minimum Acceptable
 8992 Value quantity. However, an application wishing to avail itself of the full amount of a resource
 8993 available on an implementation may make use of the value given in **<limits.h>** on that
 8994 particular implementation, by using the macros and symbolic constants listed below. It should
 8995 be noted, however, that many of the listed limits are not invariant, and at runtime, the value of
 8996 the limit may differ from those given in this header, for the following reasons:

8997 The limit is pathname-dependent.

8998 The limit differs between the compile and runtime machines.

8999 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to
 9000 determine the actual value of a limit at runtime.

9001 The items in the list ending in **_MIN** give the most negative values that the mathematical types
 9002 are guaranteed to be capable of representing. Numbers of a more negative value may be
 9003 supported on some implementations, as indicated by the **<limits.h>** header on the
 9004 implementation, but applications requiring such numbers are not guaranteed to be portable to
 9005 all implementations. For positive constants ending in **_MIN**, this indicates the minimum
 9006 acceptable value.

9007 **Runtime Invariant Values (Possibly Indeterminate)**

9008 A definition of one of the symbolic constants in the following list shall be omitted from
 9009 **<limits.h>** on specific implementations where the corresponding value is equal to or greater
 9010 than the stated minimum, but is unspecified.

9011 This indetermination might depend on the amount of available memory space on a specific
 9012 instance of a specific implementation. The actual value supported by a specific instance shall be
 9013 provided by the *sysconf()* function.

9014 {AIO_LISTIO_MAX}

9015 Maximum number of I/O operations in a single list I/O call supported by the
 9016 implementation.

9017 Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}

9018 {AIO_MAX}
 9019 Maximum number of outstanding asynchronous I/O operations supported by the
 9020 implementation.
 9021 Minimum Acceptable Value: {_POSIX_AIO_MAX}

9022 {AIO_PRIO_DELTA_MAX}
 9023 The maximum amount by which a process can decrease its asynchronous I/O priority level
 9024 from its own scheduling priority.
 9025 Minimum Acceptable Value: 0

9026 {ARG_MAX}
 9027 Maximum length of argument to the *exec* functions including environment data.
 9028 Minimum Acceptable Value: {_POSIX_ARG_MAX}

9029 {ATEXIT_MAX}
 9030 Maximum number of functions that may be registered with *atexit()*.
 9031 Minimum Acceptable Value: 32

9032 {CHILD_MAX}
 9033 Maximum number of simultaneous processes per real user ID.
 9034 Minimum Acceptable Value: {_POSIX_CHILD_MAX}

9035 {DELAYTIMER_MAX}
 9036 Maximum number of timer expiration overruns.
 9037 Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}

9038 {HOST_NAME_MAX}
 9039 Maximum length of a host name (not including the terminating null) as returned from the
 9040 *gethostname()* function.
 9041 Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}

9042 XSI {IOV_MAX}
 9043 Maximum number of *iovec* structures that one process has available for use with *readv()* or
 9044 *writev()*.
 9045 Minimum Acceptable Value: {_XOPEN_IOV_MAX}

9046 {LOGIN_NAME_MAX}
 9047 Maximum length of a login name.
 9048 Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}

9049 MSG {MQ_OPEN_MAX}
 9050 The maximum number of open message queue descriptors a process may hold.
 9051 Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}

9052 MSG {MQ_PRIO_MAX}
 9053 The maximum number of message priorities supported by the implementation.
 9054 Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}

9055 {OPEN_MAX}
 9056 A value one greater than the maximum value that the system may assign to a newly-created
 9057 file descriptor.
 9058 Minimum Acceptable Value: {_POSIX_OPEN_MAX}

9059 {PAGESIZE}
 9060 Size in bytes of a page.
 9061 Minimum Acceptable Value: 1

9062 XSI **{PAGE_SIZE}**
9063 Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
9064 defined with the same value.

9065 {PTHREAD_DESTRUCTOR_ITERATIONS}
9066 Maximum number of attempts made to destroy a thread's thread-specific data values on
9067 thread exit.
9068 Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}

9069 {PTHREAD_KEYS_MAX}
9070 Maximum number of data keys that can be created by a process.
9071 Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}

9072 {PTHREAD_STACK_MIN}
9073 Minimum size in bytes of thread stack storage.
9074 Minimum Acceptable Value: 0

9075 {PTHREAD_THREADS_MAX}
9076 Maximum number of threads that can be created per process.
9077 Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}

9078 {RTSIG_MAX}
9079 Maximum number of realtime signals reserved for application use in this implementation.
9080 Minimum Acceptable Value: {_POSIX_RTSIG_MAX}

9081 {SEM_NSEMS_MAX}
9082 Maximum number of semaphores that a process may have.
9083 Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}

9084 {SEM_VALUE_MAX}
9085 The maximum value a semaphore may have.
9086 Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}

9087 {SIGQUEUE_MAX}
9088 Maximum number of queued signals that a process may send and have pending at the
9089 receiver(s) at any time.
9090 Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}

9091 SSI | TSP **{SS_REPL_MAX}**
9092 The maximum number of replenishment operations that may be simultaneously pending
9093 for a particular sporadic server scheduler.
9094 Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}

9095 {STREAM_MAX}
9096 Maximum number of streams that one process can have open at one time. If defined, it has
9097 the same value as {FOPEN_MAX} (see <stdio.h>).
9098 Minimum Acceptable Value: {_POSIX_STREAM_MAX}

9099 {SYMLOOP_MAX}
9100 Maximum number of symbolic links that can be reliably traversed in the resolution of a
9101 pathname in the absence of a loop.
9102 Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}

9103 {TIMER_MAX}
9104 Maximum number of timers per process supported by the implementation.
9105 Minimum Acceptable Value: {_POSIX_TIMER_MAX}

9106 OB TRC **{TRACE_EVENT_NAME_MAX}**
 9107 Maximum length of the trace event name (not including the terminating null).
 9108 Minimum Acceptable Value: `{_POSIX_TRACE_EVENT_NAME_MAX}`

9109 OB TRC **{TRACE_NAME_MAX}**
 9110 Maximum length of the trace generation version string or of the trace stream name (not
 9111 including the terminating null).
 9112 Minimum Acceptable Value: `{_POSIX_TRACE_NAME_MAX}`

9113 OB TRC **{TRACE_SYS_MAX}**
 9114 Maximum number of trace streams that may simultaneously exist in the system.
 9115 Minimum Acceptable Value: `{_POSIX_TRACE_SYS_MAX}`

9116 OB TRC **{TRACE_USER_EVENT_MAX}**
 9117 Maximum number of user trace event type identifiers that may simultaneously exist in a
 9118 traced process, including the predefined user trace event
 9119 POSIX_TRACE_UNNAMED_USER_EVENT.
 9120 Minimum Acceptable Value: `{_POSIX_TRACE_USER_EVENT_MAX}`

9121 **{TTY_NAME_MAX}**
 9122 Maximum length of terminal device name.
 9123 Minimum Acceptable Value: `{_POSIX_TTY_NAME_MAX}`

9124 **{TZNAME_MAX}**
 9125 Maximum number of bytes supported for the name of a timezone (not of the *TZ* variable).
 9126 Minimum Acceptable Value: `{_POSIX_TZNAME_MAX}`

9127 **Note:** The length given by `{TZNAME_MAX}` does not include the quoting characters mentioned in
 9128 Section 8.3 (on page 177).

9129 **Pathname Variable Values**

9130 The values in the following list may be constants within an implementation or may vary from
 9131 one pathname to another. For example, file systems or directories may have different
 9132 characteristics.

9133 A definition of one of the symbolic constants in the following list shall be omitted from the
 9134 <limits.h> header on specific implementations where the corresponding value is equal to or
 9135 greater than the stated minimum, but where the value can vary depending on the file to which it
 9136 is applied. The actual value supported for a specific pathname shall be provided by the
 9137 *pathconf()* function.

9138 **{FILESIZEBITS}**
 9139 Minimum number of bits needed to represent, as a signed integer value, the maximum size
 9140 of a regular file allowed in the specified directory.
 9141 Minimum Acceptable Value: 32

9142 **{LINK_MAX}**
 9143 Maximum number of links to a single file.
 9144 Minimum Acceptable Value: `{_POSIX_LINK_MAX}`

9145 **{MAX_CANON}**
 9146 Maximum number of bytes in a terminal canonical input line.
 9147 Minimum Acceptable Value: `{_POSIX_MAX_CANON}`

9148 **{MAX_INPUT}**
 9149 Minimum number of bytes for which space is available in a terminal input queue; therefore,
 9150 the maximum number of bytes a conforming application may require to be typed as input

9151 before reading them.
 9152 Minimum Acceptable Value: {_POSIX_MAX_INPUT}

9153 {NAME_MAX}
 9154 Maximum number of bytes in a filename (not including the terminating null of a filename
 9155 string).
 9156 Minimum Acceptable Value: {_POSIX_NAME_MAX}

9157 XSI {XOPEN_NAME_MAX}

9158 {PATH_MAX}
 9159 Maximum number of bytes the implementation will store as a pathname in a user-supplied
 9160 buffer of unspecified size, including the terminating null character. Minimum number the
 9161 implementation will accept as the maximum number of bytes in a pathname.
 9162 Minimum Acceptable Value: {_POSIX_PATH_MAX}

9163 XSI {XOPEN_PATH_MAX}

9164 {PIPE_BUF}
 9165 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 9166 Minimum Acceptable Value: {_POSIX_PIPE_BUF}

9167 ADV {POSIX_ALLOC_SIZE_MIN}

9168 Minimum number of bytes of storage actually allocated for any portion of a file.
 9169 Minimum Acceptable Value: Not specified.

9170 ADV {POSIX_REC_INCR_XFER_SIZE}

9171 Recommended increment for file transfer sizes between the
 9172 {POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
 9173 Minimum Acceptable Value: Not specified.

9174 ADV {POSIX_REC_MAX_XFER_SIZE}

9175 Maximum recommended file transfer size.
 9176 Minimum Acceptable Value: Not specified.

9177 ADV {POSIX_REC_MIN_XFER_SIZE}

9178 Minimum recommended file transfer size.
 9179 Minimum Acceptable Value: Not specified.

9180 ADV {POSIX_REC_XFER_ALIGN}

9181 Recommended file transfer buffer alignment.
 9182 Minimum Acceptable Value: Not specified.

9183 {SYMLINK_MAX}
 9184 Maximum number of bytes in a symbolic link.
 9185 Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

Runtime Inceasable Values

9186
 9187 The magnitude limitations in the following list shall be fixed by specific implementations. An
 9188 application should assume that the value of the symbolic constant defined by **<limits.h>** in a
 9189 specific implementation is the minimum that pertains whenever the application is run under
 9190 that implementation. A specific instance of a specific implementation may increase the value
 9191 relative to that supplied by **<limits.h>** for that implementation. The actual value supported by a
 9192 specific instance shall be provided by the *sysconf()* function.

9193 {BC_BASE_MAX}
 9194 Maximum *obase* values allowed by the *bc* utility.
 9195 Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}

9196 {BC_DIM_MAX}
 9197 Maximum number of elements permitted in an array by the *bc* utility.
 9198 Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}

9199 {BC_SCALE_MAX}
 9200 Maximum *scale* value allowed by the *bc* utility.
 9201 Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

9202 {BC_STRING_MAX}
 9203 Maximum length of a string constant accepted by the *bc* utility.
 9204 Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

9205 {CHARCLASS_NAME_MAX}
 9206 Maximum number of bytes in a character class name.
 9207 Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

9208 {COLL_WEIGHTS_MAX}
 9209 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
 9210 keyword in the locale definition file; see [Chapter 7](#) (on page 135).
 9211 Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

9212 {EXPR_NEST_MAX}
 9213 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
 9214 Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

9215 {LINE_MAX}
 9216 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
 9217 standard input or another file), when the utility is described as processing text files. The
 9218 length includes room for the trailing <newline>.
 9219 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9220 {NGROUPS_MAX}
 9221 Maximum number of simultaneous supplementary group IDs per process.
 9222 Minimum Acceptable Value: {_POSIX2_NGROUPS_MAX}

9223 {RE_DUP_MAX}
 9224 Maximum number of repeated occurrences of a BRE or ERE interval expression; see [Section](#)
 9225 [9.3.6](#) (on page 186) and [Section 9.4.6](#) (on page 190).
 9226 Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

9227 **Maximum Values**

9228 The <limits.h> header shall define the following symbolic constants with the values shown.
 9229 These are the most restrictive values for certain features on an implementation. A conforming
 9230 implementation shall provide values no larger than these values. A conforming application must
 9231 not require a smaller value for correct operation.

9232 {_POSIX_CLOCKRES_MIN}
 9233 The resolution of the CLOCK_REALTIME clock, in nanoseconds.
 9234 Value: 20 000 000

9235 MON If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
 9236 clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

9237 **Minimum Values**

9238 The **<limits.h>** header shall define the following symbolic constants with the values shown.
9239 These are the most restrictive values for certain features on an implementation conforming to
9240 this volume of POSIX.1-2017. Related symbolic constants are defined elsewhere in this volume
9241 of POSIX.1-2017 which reflect the actual implementation and which need not be as restrictive.
9242 For each of these limits, a conforming implementation shall provide a value at least this large or
9243 shall have no limit. A strictly conforming application must not require a larger value for correct
9244 operation.

9245 {**_POSIX_AIO_LISTIO_MAX**}

9246 The number of I/O operations that can be specified in a list I/O call.
9247 Value: 2

9248 {**_POSIX_AIO_MAX**}

9249 The number of outstanding asynchronous I/O operations.
9250 Value: 1

9251 {**_POSIX_ARG_MAX**}

9252 Maximum length of argument to the *exec* functions including environment data.
9253 Value: 4 096

9254 {**_POSIX_CHILD_MAX**}

9255 Maximum number of simultaneous processes per real user ID.
9256 Value: 25

9257 {**_POSIX_DELAYTIMER_MAX**}

9258 The number of timer expiration overruns.
9259 Value: 32

9260 {**_POSIX_HOST_NAME_MAX**}

9261 Maximum length of a host name (not including the terminating null) as returned from the
9262 *gethostname()* function.
9263 Value: 255

9264 {**_POSIX_LINK_MAX**}

9265 Maximum number of links to a single file.
9266 Value: 8

9267 {**_POSIX_LOGIN_NAME_MAX**}

9268 The size of the storage required for a login name, in bytes (including the terminating null).
9269 Value: 9

9270 {**_POSIX_MAX_CANON**}

9271 Maximum number of bytes in a terminal canonical input queue.
9272 Value: 255

9273 {**_POSIX_MAX_INPUT**}

9274 Maximum number of bytes allowed in a terminal input queue.
9275 Value: 255

9276 MSG {**_POSIX_MQ_OPEN_MAX**}

9277 The number of message queues that can be open for a single process.
9278 Value: 8

9279 MSG {**_POSIX_MQ_PRIO_MAX**}

9280 The maximum number of message priorities supported by the implementation.
9281 Value: 32

9282 { _POSIX_NAME_MAX}
 9283 Maximum number of bytes in a filename (not including the terminating null of a filename
 9284 string).
 9285 Value: 14

9286 { _POSIX_NGROUPS_MAX}
 9287 Maximum number of simultaneous supplementary group IDs per process.
 9288 Value: 8

9289 { _POSIX_OPEN_MAX}
 9290 A value one greater than the maximum value that the system may assign to a newly-created
 9291 file descriptor.
 9292 Value: 20

9293 { _POSIX_PATH_MAX}
 9294 Minimum number the implementation will accept as the maximum number of bytes in a
 9295 pathname.
 9296 Value: 256

9297 { _POSIX_PIPE_BUF}
 9298 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
 9299 Value: 512

9300 { _POSIX_RE_DUP_MAX}
 9301 Maximum number of repeated occurrences of a BRE or ERE interval expression; see [Section](#)
 9302 [9.3.6](#) (on page 186) and [Section 9.4.6](#) (on page 190).
 9303 Value: 255

9304 { _POSIX_RTSIG_MAX}
 9305 The number of realtime signal numbers reserved for application use.
 9306 Value: 8

9307 { _POSIX_SEM_NSEMS_MAX}
 9308 The number of semaphores that a process may have.
 9309 Value: 256

9310 { _POSIX_SEM_VALUE_MAX}
 9311 The maximum value a semaphore may have.
 9312 Value: 32 767

9313 { _POSIX_SIGQUEUE_MAX}
 9314 The number of queued signals that a process may send and have pending at the receiver(s)
 9315 at any time.
 9316 Value: 32

9317 { _POSIX_SSIZE_MAX}
 9318 The value that can be stored in an object of type **ssize_t**.
 9319 Value: 32 767

9320 SS | TSP { _POSIX_SS_REPL_MAX}
 9321 The number of replenishment operations that may be simultaneously pending for a
 9322 particular sporadic server scheduler.
 9323 Value: 4

9324 { _POSIX_STREAM_MAX}
 9325 The number of streams that one process can have open at one time.
 9326 Value: 8

9327 { _POSIX_SYMLINK_MAX}
9328 The number of bytes in a symbolic link.
9329 Value: 255

9330 { _POSIX_SYMLINK_MAX}
9331 The number of symbolic links that can be traversed in the resolution of a pathname in the
9332 absence of a loop.
9333 Value: 8

9334 { _POSIX_THREAD_DESTRUCTOR_ITERATIONS}
9335 The number of attempts made to destroy a thread's thread-specific data values on thread
9336 exit.
9337 Value: 4

9338 { _POSIX_THREAD_KEYS_MAX}
9339 The number of data keys per process.
9340 Value: 128

9341 { _POSIX_THREAD_THREADS_MAX}
9342 The number of threads per process.
9343 Value: 64

9344 { _POSIX_TIMER_MAX}
9345 The per-process number of timers.
9346 Value: 32

9347 OB TRC { _POSIX_TRACE_EVENT_NAME_MAX}
9348 The length in bytes of a trace event name (not including the terminating null).
9349 Value: 30

9350 OB TRC { _POSIX_TRACE_NAME_MAX}
9351 The length in bytes of a trace generation version string or a trace stream name (not
9352 including the terminating null).
9353 Value: 8

9354 OB TRC { _POSIX_TRACE_SYS_MAX}
9355 The number of trace streams that may simultaneously exist in the system.
9356 Value: 8

9357 OB TRC { _POSIX_TRACE_USER_EVENT_MAX}
9358 The number of user trace event type identifiers that may simultaneously exist in a traced
9359 process, including the predefined user trace event
9360 _POSIX_TRACE_UNNAMED_USER_EVENT.
9361 Value: 32

9362 { _POSIX_TTY_NAME_MAX}
9363 The size of the storage required for a terminal device name, in bytes (including the
9364 terminating null).
9365 Value: 9

9366 { _POSIX_TZNAME_MAX}
9367 Maximum number of bytes supported for the name of a timezone (not of the TZ variable).
9368 Value: 6

9369 **Note:** The length given by { _POSIX_TZNAME_MAX} does not include the quoting characters
9370 mentioned in [Section 8.3](#) (on page 177).

9371 { _POSIX2_BC_BASE_MAX}
 9372 Maximum *obase* values allowed by the *bc* utility.
 9373 Value: 99

9374 { _POSIX2_BC_DIM_MAX}
 9375 Maximum number of elements permitted in an array by the *bc* utility.
 9376 Value: 2 048

9377 { _POSIX2_BC_SCALE_MAX}
 9378 Maximum *scale* value allowed by the *bc* utility.
 9379 Value: 99

9380 { _POSIX2_BC_STRING_MAX}
 9381 Maximum length of a string constant accepted by the *bc* utility.
 9382 Value: 1 000

9383 { _POSIX2_CHARCLASS_NAME_MAX}
 9384 Maximum number of bytes in a character class name.
 9385 Value: 14

9386 { _POSIX2_COLL_WEIGHTS_MAX}
 9387 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
 9388 keyword in the locale definition file; see [Chapter 7](#) (on page 135).
 9389 Value: 2

9390 { _POSIX2_EXPR_NEST_MAX}
 9391 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
 9392 Value: 32

9393 { _POSIX2_LINE_MAX}
 9394 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
 9395 standard input or another file), when the utility is described as processing text files. The
 9396 length includes room for the trailing <newline>.
 9397 Value: 2 048

9398 { _POSIX2_RE_DUP_MAX}
 9399 Maximum number of repeated occurrences of a BRE or ERE interval expression; see [Section](#)
 9400 [9.3.6](#) (on page 186) and [Section 9.4.6](#) (on page 190).
 9401 Value: 255

9402 XSI { _XOPEN_IOV_MAX}
 9403 Maximum number of **iovec** structures that one process has available for use with *readv()* or
 9404 *writev()*.
 9405 Value: 16

9406 XSI { _XOPEN_NAME_MAX}
 9407 Maximum number of bytes in a filename (not including the terminating null of a filename
 9408 string).
 9409 Value: 255

9410 XSI { _XOPEN_PATH_MAX}
 9411 Minimum number the implementation will accept as the maximum number of bytes in a
 9412 pathname.
 9413 Value: 1 024

9414 **Numerical Limits**

9415 The **<limits.h>** header shall define the following macros and, except for {CHAR_BIT},
9416 {LONG_BIT}, {MB_LEN_MAX}, and {WORD_BIT}, they shall be replaced by expressions that
9417 have the same type as would an expression that is an object of the corresponding type converted
9418 according to the integer promotions.

9419 If the value of an object of type **char** is treated as a signed integer when used in an expression,
9420 the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
9421 is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
9422 of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

9423 {CHAR_BIT}
9424 Number of bits in a type **char**.

9425 CX Value: 8

9426 {CHAR_MAX}
9427 Maximum value for an object of type **char**.
9428 Value: {UCHAR_MAX} or {SCHAR_MAX}

9429 {CHAR_MIN}
9430 Minimum value for an object of type **char**.
9431 Value: {SCHAR_MIN} or 0

9432 {INT_MAX}
9433 Maximum value for an object of type **int**.

9434 CX Minimum Acceptable Value: 2 147 483 647

9435 {INT_MIN}
9436 Minimum value for an object of type **int**.

9437 CX Maximum Acceptable Value: -2 147 483 647

9438 {LLONG_MAX}
9439 Maximum value for an object of type **long long**.
9440 Minimum Acceptable Value: +9 223 372 036 854 775 807

9441 {LLONG_MIN}
9442 Minimum value for an object of type **long long**.
9443 Maximum Acceptable Value: -9 223 372 036 854 775 807

9444 CX {LONG_BIT}
9445 Number of bits in an object of type **long**.
9446 Minimum Acceptable Value: 32

9447 {LONG_MAX}
9448 Maximum value for an object of type **long**.
9449 Minimum Acceptable Value: +2 147 483 647

9450 {LONG_MIN}
9451 Minimum value for an object of type **long**.
9452 Maximum Acceptable Value: -2 147 483 647

9453 {MB_LEN_MAX}
9454 Maximum number of bytes in a character, for any supported locale.
9455 Minimum Acceptable Value: 1

9456 {SCHAR_MAX}
9457 Maximum value for an object of type **signed char**.

```

9458 CX      Value: +127
9459      {SCHAR_MIN}
9460      Minimum value for an object of type signed char.
9461 CX      Value: -128
9462      {SHRT_MAX}
9463      Maximum value for an object of type short.
9464      Minimum Acceptable Value: +32 767
9465      {SHRT_MIN}
9466      Minimum value for an object of type short.
9467      Maximum Acceptable Value: -32 767
9468 CX      {SSIZE_MAX}
9469      Maximum value for an object of type ssize_t.
9470      Minimum Acceptable Value: {_POSIX_SSIZE_MAX}
9471      {UCHAR_MAX}
9472      Maximum value for an object of type unsigned char.
9473 CX      Value: 255
9474      {UINT_MAX}
9475      Maximum value for an object of type unsigned.
9476 CX      Minimum Acceptable Value: 4 294 967 295
9477      {ULLONG_MAX}
9478      Maximum value for an object of type unsigned long long.
9479      Minimum Acceptable Value: 18 446 744 073 709 551 615
9480      {ULONG_MAX}
9481      Maximum value for an object of type unsigned long.
9482      Minimum Acceptable Value: 4 294 967 295
9483      {USHRT_MAX}
9484      Maximum value for an object of type unsigned short.
9485      Minimum Acceptable Value: 65 535
9486 CX      {WORD_BIT}
9487      Number of bits in an object of type int.
9488      Minimum Acceptable Value: 32
9489
9489      Other Invariant Values
9490      The <limits.h> header shall define the following symbolic constants:
9491      {NL_ARGMAX}
9492      Maximum value of n in conversion specifications using the "%n$" sequence in calls to the
9493      printf() and scanf() families of functions.
9494      Minimum Acceptable Value: 9
9495 XSI     {NL_LANGMAX}
9496      Maximum number of bytes in a LANG name.
9497      Minimum Acceptable Value: 14
9498      {NL_MSGMAX}
9499      Maximum message number.
9500      Minimum Acceptable Value: 32 767

```


9501 {NL_SETMAX}
9502 Maximum set number.
9503 Minimum Acceptable Value: 255

9504 {NL_TEXTMAX}
9505 Maximum number of bytes in a message string.
9506 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9507 XSI {NZERO}
9508 Default process priority.
9509 Minimum Acceptable Value: 20

9510 APPLICATION USAGE

9511 None.

9512 RATIONALE

9513 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
9514 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
9515 for several reasons:

9516 They wanted to avoid making any changes to the standard that could break conforming
9517 applications, and the requested change could have that effect.

9518 The use of multiple hard links to a file cannot always be replaced with use of symbolic
9519 links. Symbolic links are semantically different from hard links in that they associate a
9520 pathname with another pathname rather than a pathname with a file. This has
9521 implications for access control, file permanence, and transparency.

9522 The original standard developers had considered the issue of allowing for
9523 implementations that did not in general support hard links, and decided that this would
9524 reduce consensus on the standard.

9525 Systems that support historical versions of the development option of the ISO POSIX-2 standard
9526 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9527 {PATH_MAX}
9528 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
9529 definition of pathname and the description of {PATH_MAX}, allowing application
9530 developers to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has
9531 been removed by correction to the {PATH_MAX} definition to include the null character.
9532 With this change, applications that previously allocated {PATH_MAX} bytes will continue to
9533 succeed.

9534 {SYMLINK_MAX}
9535 This symbol refers to space for data that is stored in the file system, as opposed to
9536 {PATH_MAX} which is the length of a name that can be passed to a function. In some
9537 existing implementations, the pathnames pointed to by symbolic links are stored in the
9538 *inodes* of the links, so it is important that {SYMLINK_MAX} not be constrained to be as large
9539 as {PATH_MAX}.

9540 FUTURE DIRECTIONS

9541 None.

9542 SEE ALSO

9543 Chapter 7 (on page 135), [<stdio.h>](#), [<unistd.h>](#)

9544 XSH Section 2.2 (on page 472), [fpathconf\(\)](#), [sysconf\(\)](#)

9545 **CHANGE HISTORY**

9546 First released in Issue 1.

9547 **Issue 5**9548 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
9549 Threads Extension.

9550 {FILESIZEBITS} is added for the Large File Summit extensions.

9551 The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to
9552 make 32-bit values the minimum requirement.

9553 The entry is restructured to improve readability.

9554 **Issue 6**9555 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN},
9556 {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum
9557 acceptable values.9558 The following new requirements on POSIX implementations derive from alignment with the
9559 Single UNIX Specification:

9560 The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.

9561 The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.

9562 The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

9563 Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX},
9564 {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

9565 The following values are added for alignment with IEEE Std 1003.1d-1999:

9566 {_POSIX_SS_REPL_MAX}

9567 {SS_REPL_MAX}

9568 {POSIX_ALLOC_SIZE_MIN}

9569 {POSIX_REC_INCR_XFER_SIZE}

9570 {POSIX_REC_MAX_XFER_SIZE}

9571 {POSIX_REC_MIN_XFER_SIZE}

9572 {POSIX_REC_XFER_ALIGN}

9573 Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN}
9574 for alignment with IEEE Std 1003.1j-2000.9575 The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment
9576 with the ISO/IEC 9899:1999 standard.

9577 The following values are added for alignment with IEEE Std 1003.1q-2000:

9578 {_POSIX_TRACE_EVENT_NAME_MAX}

9579 {_POSIX_TRACE_NAME_MAX}

9580 {_POSIX_TRACE_SYS_MAX}

9581 {_POSIX_TRACE_USER_EVENT_MAX}

9582 {TRACE_EVENT_NAME_MAX}

9583 {TRACE_NAME_MAX}

9584 {TRACE_SYS_MAX}

9585 {TRACE_USER_EVENT_MAX}

9586 The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum

9587 values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.

9588 The LEGACY symbols {PASS_MAX} and {TMP_MAX} are removed.

9589 The values for the limits {CHAR_BIT}, {SCHAR_MAX}, and {UCHAR_MAX} are now required
9590 to be 8, +127, and 255, respectively.

9591 The value for the limit {CHAR_MAX} is now {UCHAR_MAX} or {SCHAR_MAX}.

9592 The value for the limit {CHAR_MIN} is now {SCHAR_MIN} or zero.

9593 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of
9594 {_POSIX_CHILD_MAX} from 6 to 25. This is for FIPS 151-2 alignment.

9595 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for
9596 {INT_MAX}, {UINT_MAX}, and {INT_MIN} to be CX extensions over the ISO C standard, and
9597 correcting {WORD_BIT} from 16 to 32.

9598 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing
9599 {CHARCLASS_NAME_MAX} from the “Other Invariant Values” section (it also occurs under
9600 “Runtime Increaseable Values”).

9601 **Issue 7**

9602 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.

9603 Austin Group Interpretation 1003.1-2001 #173 is applied, updating the descriptions of
9604 {TRACE_EVENT_NAME_MAX} and {TRACE_NAME_MAX} to not include the terminating
9605 null.

9606 SD5-XBD-ERN-36 is applied, changing the description of {RE_DUP_MAX}.

9607 SD5-XBD-ERN-90 is applied.

9608 {NL_NMAX} is removed; it should have been removed in Issue 6.

9609 The Trace option values are marked obsolescent.

9610 The {ATEXIT_MAX}, {LONG_BIT}, {NL_MSGMAX}, {NL_SETMAX}, {NL_TEXTMAX}, and
9611 {WORD_BIT} values are moved from the XSI option to the Base.

9612 The AIO_* and _POSIX_AIO_* values are moved from the Asynchronous Input and Output
9613 option to the Base.

9614 The {_POSIX_RT_SIG_MAX}, {_POSIX_SIGQUEUE_MAX}, {RTSIG_MAX}, and
9615 {SIGQUEUE_MAX} values are moved from the Realtime Signals Extension option to the Base.

9616 Functionality relating to the Threads and Timers options is moved to the Base.

9617 This reference page is clarified with respect to macros and symbolic constants.

9618 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0052 [108], XBD/TC1-2008/0053 [291],
9619 XBD/TC1-2008/0054 [182,427], XBD/TC1-2008/0055 [291], XBD/TC1-2008/0056 [371],
9620 XBD/TC1-2008/0057 [291], XBD/TC1-2008/0058 [108], and XBD/TC1-2008/0059 [291] are
9621 applied.

9622 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0061 [666] is applied.

9623 **NAME**

9624 locale.h — category macros

9625 **SYNOPSIS**

9626 #include <locale.h>

9627 **DESCRIPTION**

9628 CX Some of the functionality described on this reference page extends the ISO C standard.
 9629 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 9630 enable the visibility of these symbols in this header.

9631 The <locale.h> header shall define the **Iconv** structure, which shall include at least the following
 9632 members. (See the definitions of *LC_MONETARY* in Section 7.3.3 (on page 155) and Section 7.3.4
 9633 (on page 158).)

- 9634 char *currency_symbol
- 9635 char *decimal_point
- 9636 char frac_digits
- 9637 char *grouping
- 9638 char *int_curr_symbol
- 9639 char int_frac_digits
- 9640 char int_n_cs_precedes
- 9641 char int_n_sep_by_space
- 9642 char int_n_sign_posn
- 9643 char int_p_cs_precedes
- 9644 char int_p_sep_by_space
- 9645 char int_p_sign_posn
- 9646 char *mon_decimal_point
- 9647 char *mon_grouping
- 9648 char *mon_thousands_sep
- 9649 char *negative_sign
- 9650 char n_cs_precedes
- 9651 char n_sep_by_space
- 9652 char n_sign_posn
- 9653 char *positive_sign
- 9654 char p_cs_precedes
- 9655 char p_sep_by_space
- 9656 char p_sign_posn
- 9657 char *thousands_sep

9658 The <locale.h> header shall define NULL (as described in <stddef.h>) and at least the following
 9659 as macros:

- 9660 LC_ALL
- 9661 LC_COLLATE
- 9662 LC_CTYPE
- 9663 CX LC_MESSAGES
- 9664 LC_MONETARY
- 9665 LC_NUMERIC
- 9666 LC_TIME

9667 which shall expand to integer constant expressions with distinct values for use as the first
 9668 argument to the *setlocale()* function.

9669 Additional macro definitions, beginning with the characters *LC_* and an uppercase letter, may
 9670 also be specified by the implementation.

9671 CX The **<locale.h>** header shall contain at least the following macros representing bitmasks for use
 9672 with the *newlocale()* function for each supported locale category:

9673 LC_COLLATE_MASK
 9674 LC_CTYPE_MASK
 9675 LC_MESSAGES_MASK
 9676 LC_MONETARY_MASK
 9677 LC_NUMERIC_MASK
 9678 LC_TIME_MASK

9679 In addition, a macro to set the bits for all categories set shall be defined:

9680 LC_ALL_MASK

9681 The **<locale.h>** header shall define LC_GLOBAL_LOCALE, a special locale object descriptor
 9682 used by the *duplocale()* and *uselocale()* functions.

9683 The **<locale.h>** header shall define the **locale_t** type, representing a locale object.

9684 The following shall be declared as functions and may also be defined as macros. Function
 9685 prototypes shall be provided for use with ISO C standard compilers.

9686 CX locale_t duplocale(locale_t);
 9687 void freelocale(locale_t);
 9688 struct lconv *localeconv(void);
 9689 CX locale_t newlocale(int, const char *, locale_t);
 9690 char *setlocale(int, const char *);
 9691 CX locale_t uselocale (locale_t);

9692 APPLICATION USAGE

9693 None.

9694 RATIONALE

9695 It is suggested that each category macro name for use in *setlocale()* have a corresponding macro
 9696 name ending in *_MASK* for use in *newlocale()*.

9697 FUTURE DIRECTIONS

9698 None.

9699 SEE ALSO

9700 [Chapter 8](#) (on page 173), **<stddef.h>**

9701 XSH *duplocale()*, *freelocale()*, *localeconv()*, *newlocale()*, *setlocale()*, *uselocale()*

9702 CHANGE HISTORY

9703 First released in Issue 3.

9704 Included for alignment with the ISO C standard.

9705 Issue 6

9706 The **lconv** structure is expanded with new members (**int_n_cs_precedes**, **int_n_sep_by_space**,
 9707 **int_n_sign_posn**, **int_p_cs_precedes**, **int_p_sep_by_space**, and **int_p_sign_posn**) for alignment
 9708 with the ISO/IEC 9899:1999 standard.

9709 Extensions beyond the ISO C standard are marked.

9710 **Issue 7**

9711 The *duplocale()*, *freelocale()*, *newlocale()*, and *uselocale()* functions are added from The Open
9712 Group Technical Standard, 2006, Extended API Set Part 4.

9713 This reference page is clarified with respect to macros and symbolic constants.

9714 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0060 [301,427] is applied.

9715 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0062 [781] is applied.

9716 **NAME**9717 `math.h` ‡mathematical declarations9718 **SYNOPSIS**9719 `#include <math.h>`9720 **DESCRIPTION**

9721 CX Some of the functionality described on this reference page extends the ISO C standard.
 9722 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 9723 enable the visibility of these symbols in this header.

9724 The **<math.h>** header shall define at least the following types:9725 **float_t** A real-floating type at least as wide as **float**.9726 **double_t** A real-floating type at least as wide as **double**, and at least as wide as **float_t**.

9727 If FLT_EVAL_METHOD equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9728 FLT_EVAL_METHOD equals 1, they shall both be **double**; if FLT_EVAL_METHOD equals 2,
 9729 they shall both be **long double**; for other values of FLT_EVAL_METHOD, they are otherwise
 9730 implementation-defined.

9731 The **<math.h>** header shall define the following macros, where real-floating indicates that the
 9732 argument shall be an expression of real-floating type:

```
9733 int fpclassify(real-floating x);
9734 int isfinite(real-floating x);
9735 int isgreater(real-floating x, real-floating y);
9736 int isgreaterequal(real-floating x, real-floating y);
9737 int isinf(real-floating x);
9738 int isless(real-floating x, real-floating y);
9739 int islessequal(real-floating x, real-floating y);
9740 int islessgreater(real-floating x, real-floating y);
9741 int isnan(real-floating x);
9742 int isnormal(real-floating x);
9743 int isunordered(real-floating x, real-floating y);
9744 int signbit(real-floating x);
```

9745 The **<math.h>** header shall define the following symbolic constants. The values shall have type
 9746 **double** and shall be accurate to at least the precision of the **double** type.

9747	XSI	M_E	Value of e
9748	XSI	M_LOG2E	Value of $\log_2 e$
9749	XSI	M_LOG10E	Value of $\log_{10} e$
9750	XSI	M_LN2	Value of $\log_e 2$
9751	XSI	M_LN10	Value of $\log_e 10$
9752	XSI	M_PI	Value of π
9753	XSI	M_PI_2	Value of $\pi/2$
9754	XSI	M_PI_4	Value of $\pi/4$
9755	XSI	M_1_PI	Value of $1/\pi$
9756	XSI	M_2_PI	Value of $2/\pi$

9757	XSI	M_2_SQRTPI	Value of $2\sqrt{\pi}$
9758	XSI	M_SQRT2	Value of $\sqrt{2}$
9759	XSI	M_SQRT1_2	Value of $1/\sqrt{2}$
9760		The <math.h> header shall define the following symbolic constant:	
9761	OB XSI	MAXFLOAT	Same value as FLT_MAX in <float.h>.
9762		The <math.h> header shall define the following macros:	
9763		HUGE_VAL	A positive double constant expression, not necessarily representable as a float . Used as an error value returned by the mathematics library. HUGE_VAL evaluates to +infinity on systems supporting IEEE Std 754-1985.
9764			
9765			
9766		HUGE_VALF	A positive float constant expression. Used as an error value returned by the mathematics library. HUGE_VALF evaluates to +infinity on systems supporting IEEE Std 754-1985.
9767			
9768			
9769		HUGE_VALL	A positive long double constant expression. Used as an error value returned by the mathematics library. HUGE_VALL evaluates to +infinity on systems supporting IEEE Std 754-1985.
9770			
9771			
9772		INFINITY	A constant expression of type float representing positive or unsigned infinity, if available; else a positive constant of type float that overflows at translation time.
9773			
9774			
9775		NAN	A constant expression of type float representing a quiet NaN. This macro is only defined if the implementation supports quiet NaNs for the float type.
9776			
9777		The following macros shall be defined for number classification. They represent the mutually-exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with FP_ and an uppercase letter, may also be specified by the implementation.	
9778			
9779			
9780			
9781			
9782		FP_INFINITE	
9783		FP_NAN	
9784		FP_NORMAL	
9785		FP_SUBNORMAL	
9786		FP_ZERO	
9787		The following optional macros indicate whether the <i>fma()</i> family of functions are fast compared with direct code:	
9788			
9789		FP_FAST_FMA	
9790		FP_FAST_FMAF	
9791		FP_FAST_FMAL	
9792		If defined, the FP_FAST_FMA macro shall expand to the integer constant 1 and shall indicate that the <i>fma()</i> function generally executes about as fast as, or faster than, a multiply and an add of double operands. If undefined, the speed of execution is unspecified. The other macros have the equivalent meaning for the float and long double versions.	
9793			
9794			
9795			
9796		The following macros shall expand to integer constant expressions whose values are returned by <i>ilogb(x)</i> if <i>x</i> is zero or NaN, respectively. The value of FP_ILOGB0 shall be either { INT_MIN } or -{INT_MAX} . The value of FP_ILOGBNAN shall be either { INT_MAX } or { INT_MIN }.	
9797			
9798			

9799 FP_ILOGB0
 9800 FP_ILOGBNAN

9801 The following macros shall expand to the integer constants 1 and 2, respectively;

9802 MATH_ERRNO
 9803 MATH_ERREXCEPT

9804 The following macro shall expand to an expression that has type **int** and the value
 9805 MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both:

9806 math_errhandling

9807 The value of `math_errhandling` is constant for the duration of the program. It is unspecified
 9808 whether `math_errhandling` is a macro or an identifier with external linkage. If a macro definition
 9809 is suppressed or a program defines an identifier with the name `math_errhandling`, the behavior
 9810 is undefined. If the expression `(math_errhandling & MATH_ERREXCEPT)` can be non-zero, the
 9811 implementation shall define the macros `FE_DIVBYZERO`, `FE_INVALID`, and `FE_OVERFLOW` in
 9812 **<fenv.h>**.

9813 The following shall be declared as functions and may also be defined as macros. Function
 9814 prototypes shall be provided.

9815 double acos(double);
 9816 float acosf(float);
 9817 double acosh(double);
 9818 float acoshf(float);
 9819 long double acoshl(long double);
 9820 long double acosl(long double);
 9821 double asin(double);
 9822 float asinf(float);
 9823 double asinh(double);
 9824 float asinhf(float);
 9825 long double asinh1(long double);
 9826 long double asin1(long double);
 9827 double atan(double);
 9828 double atan2(double, double);
 9829 float atan2f(float, float);
 9830 long double atan2l(long double, long double);
 9831 float atanf(float);
 9832 double atanh(double);
 9833 float atanhf(float);
 9834 long double atanh1(long double);
 9835 long double atan1(long double);
 9836 double cbrt(double);
 9837 float cbrtf(float);
 9838 long double cbrt1(long double);
 9839 double ceil(double);
 9840 float ceilf(float);
 9841 long double ceill(long double);
 9842 double copysign(double, double);
 9843 float copysignf(float, float);
 9844 long double copysignl(long double, long double);
 9845 double cos(double);

```

9846     float      cosf(float);
9847     double     cosh(double);
9848     float      coshf(float);
9849     long double coshl(long double);
9850     long double cosl(long double);
9851     double     erf(double);
9852     double     erfc(double);
9853     float      erfcf(float);
9854     long double erfcl(long double);
9855     float      erff(float);
9856     long double erfl(long double);
9857     double     exp(double);
9858     double     exp2(double);
9859     float      exp2f(float);
9860     long double exp2l(long double);
9861     float      expf(float);
9862     long double expl(long double);
9863     double     expm1(double);
9864     float      expm1f(float);
9865     long double expm1l(long double);
9866     double     fabs(double);
9867     float      fabsf(float);
9868     long double fabsl(long double);
9869     double     fdim(double, double);
9870     float      fdimf(float, float);
9871     long double fdiml(long double, long double);
9872     double     floor(double);
9873     float      floorf(float);
9874     long double floorl(long double);
9875     double     fma(double, double, double);
9876     float      fmaf(float, float, float);
9877     long double fmal(long double, long double, long double);
9878     double     fmax(double, double);
9879     float      fmaxf(float, float);
9880     long double fmaxl(long double, long double);
9881     double     fmin(double, double);
9882     float      fminf(float, float);
9883     long double fminl(long double, long double);
9884     double     fmod(double, double);
9885     float      fmodf(float, float);
9886     long double fmodl(long double, long double);
9887     double     frexp(double, int *);
9888     float      frexpf(float, int *);
9889     long double frexpl(long double, int *);
9890     double     hypot(double, double);
9891     float      hypotf(float, float);
9892     long double hypotl(long double, long double);
9893     int        ilogb(double);
9894     int        ilogbf(float);
9895     int        ilogbl(long double);
9896     XSI       double     j0(double);
9897     double     j1(double);

```

```
9898     double      jn(int, double);
9899     double      ldexp(double, int);
9900     float       ldexpf(float, int);
9901     long double  ldexpl(long double, int);
9902     double      lgamma(double);
9903     float       lgammaf(float);
9904     long double  lgammal(long double);
9905     long long    llrint(double);
9906     long long    llrintf(float);
9907     long long    llrintl(long double);
9908     long long    llround(double);
9909     long long    llroundf(float);
9910     long long    llroundl(long double);
9911     double      log(double);
9912     double      log10(double);
9913     float       log10f(float);
9914     long double  log10l(long double);
9915     double      log1p(double);
9916     float       log1pf(float);
9917     long double  log1pl(long double);
9918     double      log2(double);
9919     float       log2f(float);
9920     long double  log2l(long double);
9921     double      logb(double);
9922     float       logbf(float);
9923     long double  logbl(long double);
9924     float       logf(float);
9925     long double  logl(long double);
9926     long        lrint(double);
9927     long        lrintf(float);
9928     long        lrintl(long double);
9929     long        lround(double);
9930     long        lroundf(float);
9931     long        lroundl(long double);
9932     double      modf(double, double *);
9933     float       modff(float, float *);
9934     long double  modfl(long double, long double *);
9935     double      nan(const char *);
9936     float       nanf(const char *);
9937     long double  nanl(const char *);
9938     double      nearbyint(double);
9939     float       nearbyintf(float);
9940     long double  nearbyintl(long double);
9941     double      nextafter(double, double);
9942     float       nextafterf(float, float);
9943     long double  nextafterl(long double, long double);
9944     double      nexttoward(double, long double);
9945     float       nexttowardf(float, long double);
9946     long double  nexttowardl(long double, long double);
9947     double      pow(double, double);
9948     float       powf(float, float);
9949     long double  powl(long double, long double);
```

```

9950     double      remainder(double, double);
9951     float       remainderf(float, float);
9952     long double  remainderl(long double, long double);
9953     double      remquo(double, double, int *);
9954     float       remquof(float, float, int *);
9955     long double  remquol(long double, long double, int *);
9956     double      rint(double);
9957     float       rintf(float);
9958     long double  rintl(long double);
9959     double      round(double);
9960     float       roundf(float);
9961     long double  roundl(long double);
9962     double      scalbn(double, long);
9963     float       scalbnf(float, long);
9964     long double  scalbnl(long double, long);
9965     double      scalbn(double, int);
9966     float       scalbnf(float, int);
9967     long double  scalbnl(long double, int);
9968     double      sin(double);
9969     float       sinf(float);
9970     double      sinh(double);
9971     float       sinhf(float);
9972     long double  sinhl(long double);
9973     long double  sinl(long double);
9974     double      sqrt(double);
9975     float       sqrtf(float);
9976     long double  sqrtl(long double);
9977     double      tan(double);
9978     float       tanf(float);
9979     double      tanh(double);
9980     float       tanhf(float);
9981     long double  tanhl(long double);
9982     long double  tanl(long double);
9983     double      tgamma(double);
9984     float       tgammaf(float);
9985     long double  tgamma_l(long double);
9986     double      trunc(double);
9987     float       truncf(float);
9988     long double  trunc_l(long double);
9989     XSI double    y0(double);
9990     double      y1(double);
9991     double      yn(int, double);

```

9992 The following external variable shall be defined:

```

9993 XSI extern int signgam;

```

9994 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces
9995 volume of POSIX.1-2017 for all representable values of its input arguments, except where stated
9996 otherwise. Each function shall execute as if it were a single operation without generating any
9997 externally visible exceptional conditions.

9998 **APPLICATION USAGE**

9999 The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is off) the implementation to contract expressions. Each pragma can occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement.
 10000
 10001 When outside external declarations, the pragma takes effect from its occurrence until another
 10002 FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a
 10003 compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT
 10004 pragma is encountered (including within a nested compound statement), or until the end of the
 10005 compound statement; at the end of a compound statement the state for the pragma is restored to
 10006 its condition just before the compound statement. If this pragma is used in any other context, the
 10007 behavior is undefined. The default state (on or off) for the pragma is implementation-defined.
 10008

10009 Applications should use FLT_MAX as described in the **<float.h>** header instead of the
 10010 obsolescent MAXFLOAT.

10011 Note that if FLT_EVAL_METHOD is neither 0 nor 1, then some constants might not compare
 10012 equal as expected; for example, (double)M_PI == M_PI can fail.

10013 **RATIONALE**

10014 Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type
 10015 **double**. All the names formed by appending 'f' or 'l' to a name in **<math.h>** were reserved
 10016 to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999
 10017 standard provides for all three versions of math functions.

10018 The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their
 10019 capability is available through *sprintf()*.

10020 **FUTURE DIRECTIONS**

10021 None.

10022 **SEE ALSO**

10023 **<float.h>**, **<stddef.h>**, **<sys/types.h>**

10024 XSH Section 2.2 (on page 472), *acos()*, *acosh()*, *asin()*, *asinh()*, *atan()*, *atan2()*, *atanh()*, *cbrt()*,
 10025 *ceil()*, *copysign()*, *cos()*, *cosh()*, *erf()*, *erfc()*, *exp()*, *exp2()*, *expm1()*, *fabs()*, *fdim()*, *floor()*, *fma()*,
 10026 *fmax()*, *fmin()*, *fmod()*, *fpclassify()*, *frexp()*, *hypot()*, *ilogb()*, *isfinite()*, *isgreater()*, *isgreaterequal()*,
 10027 *isinf()*, *isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *j0()*, *ldexp()*,
 10028 *lgamma()*, *llrint()*, *llround()*, *log()*, *log10()*, *log1p()*, *log2()*, *logb()*, *lrint()*, *lround()*, *modf()*, *nan()*,
 10029 *nearbyint()*, *nextafter()*, *pow()*, *remainder()*, *remquo()*, *rint()*, *round()*, *scalbln()*, *signbit()*, *sin()*,
 10030 *sinh()*, *sqrt()*, *tan()*, *tanh()*, *tgamma()*, *trunc()*, *y0()*

10031 **CHANGE HISTORY**

10032 First released in Issue 1.

10033 **Issue 6**

10034 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

10035 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the
 10036 meaning of the FP_FAST_FMA macro is unspecified if the macro is undefined.

10037 **Issue 7**

10038 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #47 (SD5-XBD-ERN-52) is applied,
 10039 clarifying the wording of the FP_FAST_FMA macro.

10040 The MAXFLOAT constant is marked obsolescent.

10041 This reference page is clarified with respect to macros and symbolic constants.

10042
10043

POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0063 [801] and XBD/TC2-2008/0064 [801] are applied.

10044 **NAME**

10045 monetary.h — monetary types

10046 **SYNOPSIS**

10047 #include <monetary.h>

10048 **DESCRIPTION**10049 The **<monetary.h>** header shall define the **locale_t** type as described in **<locale.h>**.10050 The **<monetary.h>** header shall define the **size_t** type as described in **<stddef.h>**.10051 The **<monetary.h>** header shall define the **ssize_t** type as described in **<sys/types.h>**.10052 The following shall be declared as functions and may also be defined as macros. Function
10053 prototypes shall be provided for use with ISO C standard compilers.

10054 ssize_t strfmon(char *restrict, size_t, const char *restrict, ...);

10055 ssize_t strfmon_l(char *restrict, size_t, locale_t,

10056 const char *restrict, ...);

10057 **APPLICATION USAGE**

10058 None.

10059 **RATIONALE**

10060 None.

10061 **FUTURE DIRECTIONS**

10062 None.

10063 **SEE ALSO**10064 [<locale.h>](#), [<stddef.h>](#), [<sys/types.h>](#)10065 XSH [strfmon\(\)](#)10066 **CHANGE HISTORY**

10067 First released in Issue 4.

10068 **Issue 6**10069 The **restrict** keyword is added to the prototype for *strfmon()*.10070 **Issue 7**10071 The **<monetary.h>** header is moved from the XSI option to the Base.10072 The *strfmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API
10073 Set Part 4.10074 A declaration for the **locale_t** type is added.

10075 **NAME**

10076 mqueue.h ‡message queues (REALTIME)

10077 **SYNOPSIS**

10078 MSG #include <mqueue.h>

10079 **DESCRIPTION**

10080 The <mqueue.h> header shall define the **mqd_t** type, which is used for message queue
10081 descriptors. This is not an array type.

10082 The <mqueue.h> header shall define the **pthread_attr_t**, **size_t**, and **ssize_t** types as described
10083 in <sys/types.h>.

10084 The <mqueue.h> header shall define the **struct timespec** structure as described in <time.h>.

10085 The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which
10086 are described in the <signal.h> header.

10087 The <mqueue.h> header shall define the **mq_attr** structure, which is used in getting and setting
10088 the attributes of a message queue. Attributes are initially set when the message queue is created.
10089 An **mq_attr** structure shall have at least the following fields:

10090	long	mq_flags	Message queue flags.
10091	long	mq_maxmsg	Maximum number of messages.
10092	long	mq_msgsize	Maximum message size.
10093	long	mq_curmsgs	Number of messages currently queued.

10094 The following shall be declared as functions and may also be defined as macros. Function
10095 prototypes shall be provided.

```

10096 int      mq_close(mqd_t);
10097 int      mq_getattr(mqd_t, struct mq_attr *);
10098 int      mq_notify(mqd_t, const struct sigevent *);
10099 mqd_t    mq_open(const char *, int, ...);
10100 ssize_t  mq_receive(mqd_t, char *, size_t, unsigned *);
10101 int      mq_send(mqd_t, const char *, size_t, unsigned);
10102 int      mq_setattr(mqd_t, const struct mq_attr *restrict,
10103                   struct mq_attr *restrict);
10104 ssize_t  mq_timedreceive(mqd_t, char *restrict, size_t,
10105                        unsigned *restrict, const struct timespec *restrict);
10106 int      mq_timedsend(mqd_t, const char *, size_t, unsigned,
10107                      const struct timespec *);
10108 int      mq_unlink(const char *);
    
```

10109 Inclusion of the <mqueue.h> header may make visible symbols defined in the headers
10110 <fcntl.h>, <signal.h>, and <time.h>.

10111 **APPLICATION USAGE**

10112 None.

10113 **RATIONALE**

10114 None.

10115 **FUTURE DIRECTIONS**

10116 None.

10117 **SEE ALSO**10118 [<fcntl.h>](#), [<signal.h>](#), [<sys/types.h>](#), [<time.h>](#)10119 XSH [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#), [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#),
10120 [mq_unlink\(\)](#)10121 **CHANGE HISTORY**

10122 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10123 **Issue 6**10124 The **<mqqueue.h>** header is marked as part of the Message Passing option.10125 The [mq_timedreceive\(\)](#) and [mq_timedsend\(\)](#) functions are added for alignment with IEEE Std
10126 1003.1d-1999.10127 The **restrict** keyword is added to the prototypes for [mq_setattr\(\)](#) and [mq_timedreceive\(\)](#).10128 **Issue 7**

10129 Type and structure declarations are added.

10130 **NAME**

10131 ndbm.h ¶ definitions for ndbm database operations

10132 **SYNOPSIS**

10133 XSI #include <ndbm.h>

10134 **DESCRIPTION**10135 The <ndbm.h> header shall define the **datum** type as a structure, which shall include at least the
10136 following members:10137 void *dptr A pointer to the application's data.
10138 size_t dsize The size of the object pointed to by *dptr*.10139 The <ndbm.h> header shall define the **size_t** type as described in <stddef.h>.10140 The <ndbm.h> header shall define the **DBM** type.10141 The <ndbm.h> header shall define the following symbolic constants as possible values for the
10142 *store_mode* argument to *dbm_store()*:

10143 DBM_INSERT Insertion of new entries only.

10144 DBM_REPLACE Allow replacing existing entries.

10145 The following shall be declared as functions and may also be defined as macros. Function
10146 prototypes shall be provided.10147 int dbm_clearerr(DBM *);
10148 void dbm_close(DBM *);
10149 int dbm_delete(DBM *, datum);
10150 int dbm_error(DBM *);
10151 datum dbm_fetch(DBM *, datum);
10152 datum dbm_firstkey(DBM *);
10153 datum dbm_nextkey(DBM *);
10154 DBM *dbm_open(const char *, int, mode_t);
10155 int dbm_store(DBM *, datum, datum, int);10156 The <ndbm.h> header shall define the **mode_t** type through **typedef**, as described in
10157 <sys/types.h>.10158 **APPLICATION USAGE**

10159 None.

10160 **RATIONALE**

10161 None.

10162 **FUTURE DIRECTIONS**

10163 None.

10164 **SEE ALSO**

10165 <stddef.h>, <sys/types.h>

10166 XSH *dbm_clearerr()*10167 **CHANGE HISTORY**

10168 First released in Issue 4, Version 2.

10169 **Issue 5**

10170 References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.

10171 **Issue 7**

10172 This reference page is clarified with respect to macros and symbolic constants.

10173 **NAME**

10174 net/if.h ‡sockets local interfaces

10175 **SYNOPSIS**

10176 #include <net/if.h>

10177 **DESCRIPTION**10178 The <net/if.h> header shall define the **if_nameindex** structure, which shall include at least the
10179 following members:10180 unsigned if_index Numeric index of the interface.
10181 char *if_name Null-terminated name of the interface.10182 The <net/if.h> header shall define the following symbolic constant for the length of a buffer
10183 containing an interface name (including the terminating NULL character):10184 **IF_NAMESIZE** Interface name length.10185 The following shall be declared as functions and may also be defined as macros. Function
10186 prototypes shall be provided.10187 void if_freenameindex(struct if_nameindex *);
10188 char *if_indextoname(unsigned, char *);
10189 struct if_nameindex *if_nameindex(void);
10190 unsigned if_nametoindex(const char *);10191 **APPLICATION USAGE**

10192 None.

10193 **RATIONALE**

10194 None.

10195 **FUTURE DIRECTIONS**

10196 None.

10197 **SEE ALSO**10198 XSH [if_freenameindex\(\)](#), [if_indextoname\(\)](#), [if_nameindex\(\)](#), [if_nametoindex\(\)](#)10199 **CHANGE HISTORY**

10200 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10201 **Issue 7**

10202 This reference page is clarified with respect to macros and symbolic constants.

10203 **NAME**10204 `netdb.h` ¶ definitions for network database operations10205 **SYNOPSIS**10206 `#include <netdb.h>`10207 **DESCRIPTION**10208 The **<netdb.h>** header may define the `in_port_t` type and the `in_addr_t` type as described in
10209 **<netinet/in.h>**.10210 The **<netdb.h>** header shall define the `hostent` structure, which shall include at least the
10211 following members:

10212	<code>char</code>	<code>*h_name</code>	Official name of the host.
10213	<code>char</code>	<code>**h_aliases</code>	A pointer to an array of pointers to 10214 alternative host names, terminated by a 10215 null pointer.
10216	<code>int</code>	<code>h_addrtype</code>	Address type.
10217	<code>int</code>	<code>h_length</code>	The length, in bytes, of the address.
10218	<code>char</code>	<code>**h_addr_list</code>	A pointer to an array of pointers to network 10219 addresses (in network byte order) for the host, 10220 terminated by a null pointer.

10221 The **<netdb.h>** header shall define the `netent` structure, which shall include at least the
10222 following members:

10223	<code>char</code>	<code>*n_name</code>	Official, fully-qualified (including the 10224 domain) name of the host.
10225	<code>char</code>	<code>**n_aliases</code>	A pointer to an array of pointers to 10226 alternative network names, terminated by a 10227 null pointer.
10228	<code>int</code>	<code>n_addrtype</code>	The address type of the network.
10229	<code>uint32_t</code>	<code>n_net</code>	The network number, in host byte order.

10230 The **<netdb.h>** header shall define the `uint32_t` type as described in **<inttypes.h>**.10231 The **<netdb.h>** header shall define the `protoent` structure, which shall include at least the
10232 following members:

10233	<code>char</code>	<code>*p_name</code>	Official name of the protocol.
10234	<code>char</code>	<code>**p_aliases</code>	A pointer to an array of pointers to 10235 alternative protocol names, terminated by 10236 a null pointer.
10237	<code>int</code>	<code>p_proto</code>	The protocol number.

10238 The **<netdb.h>** header shall define the `servent` structure, which shall include at least the
10239 following members:

10240	<code>char</code>	<code>*s_name</code>	Official name of the service.
10241	<code>char</code>	<code>**s_aliases</code>	A pointer to an array of pointers to 10242 alternative service names, terminated by 10243 a null pointer.
10244	<code>int</code>	<code>s_port</code>	A value which, when converted to <code>uint16_t</code> , 10245 yields the port number in network byte order 10246 at which the service resides.
10247	<code>char</code>	<code>*s_proto</code>	The name of the protocol to use when 10248 contacting the service.

10249 The <netdb.h> header shall define the IPPORT_RESERVED symbolic constant with the value of
 10250 the highest reserved Internet port number.

10251 **Address Information Structure**

10252 The <netdb.h> header shall define the **addrinfo** structure, which shall include at least the
 10253 following members:

10254	int	ai_flags	Input flags.
10255	int	ai_family	Address family of socket.
10256	int	ai_socktype	Socket type.
10257	int	ai_protocol	Protocol of socket.
10258	socklen_t	ai_addrlen	Length of socket address.
10259	struct sockaddr	*ai_addr	Socket address of socket.
10260	char	*ai_canonname	Canonical name of service location.
10261	struct addrinfo	*ai_next	Pointer to next in list.

10262 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-
 10263 distinct integer constants for use in the *flags* field of the **addrinfo** structure:

10264	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
10265	AI_CANONNAME	Request for canonical name.
10266	AI_NUMERICHOST	Return numeric host address as name.
10267	AI_NUMERICSERV	Inhibit service name resolution.
10268	AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them 10269 to the caller as IPv4-mapped IPv6 addresses.
10270	AI_ALL	Query for both IPv4 and IPv6 addresses.
10271	AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query 10272 for IPv6 addresses only when an IPv6 address is configured.

10273 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-
 10274 distinct integer constants for use in the *flags* argument to *getnameinfo()*:

10275	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
10276	NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.
10277	NI_NAMEREQD	Return an error if the node's name cannot be located in the database.
10278	NI_NUMERICSERV	The numeric form of the service address is returned instead of its name.
10279	NI_NUMERICSCOPE	For IPv6 addresses, the numeric form of the scope identifier is returned 10280 instead of its name.
10281		
10282	NI_DGRAM	Indicates that the service is a datagram service (SOCK_DGRAM).

10283 **Address Information Errors**

10284 The **<netdb.h>** header shall define the following symbolic constants for use as error values for
 10285 *getaddrinfo()* and *getnameinfo()*. The values shall be suitable for use in **#if** preprocessing
 10286 directives.

10287 **EAI_AGAIN** The name could not be resolved at this time. Future attempts may
 10288 succeed.

10289 **EAI_BADFLAGS** The flags had an invalid value.

10290 **EAI_FAIL** A non-recoverable error occurred.

10291 **EAI_FAMILY** The address family was not recognized or the address length was invalid
 10292 for the specified family.

10293 **EAI_MEMORY** There was a memory allocation failure.

10294 **EAI_NONAME** The name does not resolve for the supplied parameters.

10295 **NI_NAMEREQD** is set and the host's name cannot be located, or both
 10296 *nodename* and *servertime* were null.

10297 **EAI_SERVICE** The service passed was not recognized for the specified socket type.

10298 **EAI_SOCKTYPE** The intended socket type was not recognized.

10299 **EAI_SYSTEM** A system error occurred. The error code can be found in *errno*.

10300 **EAI_OVERFLOW** An argument buffer overflowed.

10301 The following shall be declared as functions and may also be defined as macros. Function
 10302 prototypes shall be provided.

```

10303 void          endhostent(void);
10304 void          endnetent(void);
10305 void          endprotoent(void);
10306 void          endservent(void);
10307 void          freeaddrinfo(struct addrinfo *);
10308 const char    *gai_strerror(int);
10309 int           getaddrinfo(const char *restrict, const char *restrict,
10310                          const struct addrinfo *restrict,
10311                          struct addrinfo **restrict);
10312 struct hostent *gethostent(void);
10313 int           getnameinfo(const struct sockaddr *restrict, socklen_t,
10314                          char *restrict, socklen_t, char *restrict,
10315                          socklen_t, int);
10316 struct netent *getnetbyaddr(uint32_t, int);
10317 struct netent *getnetbyname(const char *);
10318 struct netent *getnetent(void);
10319 struct protoent *getprotobyname(const char *);
10320 struct protoent *getprotobynumber(int);
10321 struct protoent *getprotoent(void);
10322 struct servent *getservbyname(const char *, const char *);
10323 struct servent *getservbyport(int, const char *);
10324 struct servent *getservent(void);
10325 void          sethostent(int);
10326 void          setnetent(int);
10327 void          setprotoent(int);

```

10328 void setserverent(int);

10329 The <netdb.h> header shall define the `socklen_t` type through `typedef`, as described in
10330 <sys/socket.h>.

10331 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,
10332 <sys/socket.h>, and <inttypes.h>.

10333 APPLICATION USAGE

10334 None.

10335 RATIONALE

10336 None.

10337 FUTURE DIRECTIONS

10338 None.

10339 SEE ALSO

10340 <inttypes.h>, <netinet/in.h>, <sys/socket.h>

10341 XSH *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *freeaddrinfo()*, *gai_strerror()*,
10342 *getnameinfo()*

10343 CHANGE HISTORY

10344 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10345 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for
10346 *gai_strerror()* from `char *` to `const char *`. This is for coordination with the IPnG Working Group.

10347 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the
10348 NI_NUMERICSCOPE macro and correcting the *getnameinfo()* function prototype. These changes
10349 are for alignment with IPv6.

10350 Issue 7

10351 SD5-XBD-ERN-14 is applied, changing the description of the *s_port* member of the `servent`
10352 structure.

10353 The obsolescent *h_errno* external integer, and the obsolescent *gethostbyaddr()* and *gethostbyname()*
10354 functions are removed, along with the `HOST_NOT_FOUND`, `NO_DATA`, `NO_RECOVERY`, and
10355 `TRY_AGAIN` macros.

10356 This reference page is clarified with respect to macros and symbolic constants.

10357 **NAME**

10358 netinet/in.h — Internet address family

10359 **SYNOPSIS**

10360 #include <netinet/in.h>

10361 **DESCRIPTION**10362 The **<netinet/in.h>** header shall define the following types:10363 **in_port_t** Equivalent to the type **uint16_t** as described in **<inttypes.h>**.10364 **in_addr_t** Equivalent to the type **uint32_t** as described in **<inttypes.h>**.10365 The **<netinet_in.h>** header shall define the **sa_family_t** type as described in **<sys/socket.h>**.10366 The **<netinet_in.h>** header shall define the **uint8_t** and **uint32_t** types as described in **<inttypes.h>**. Inclusion of the **<netinet/in.h>** header may also make visible all symbols from **<inttypes.h>** and **<sys/socket.h>**.10369 The **<netinet/in.h>** header shall define the **in_addr** structure, which shall include at least the following member:

10371 in_addr_t s_addr

10372 The **<netinet/in.h>** header shall define the **sockaddr_in** structure, which shall include at least the following members:

10374	sa_family_t	sin_family	AF_INET.
10375	in_port_t	sin_port	Port number.
10376	struct in_addr	sin_addr	IP address.

10377 The *sin_port* and *sin_addr* members shall be in network byte order.10378 The **sockaddr_in** structure is used to store addresses for the Internet address family. Pointers to this type shall be cast by applications to **struct sockaddr *** for use with socket functions.10380 IP6 The **<netinet/in.h>** header shall define the **in6_addr** structure, which shall include at least the following member:

10382 uint8_t s6_addr[16]

10383 This array is used to contain a 128-bit IPv6 address, stored in network byte order.

10384 The **<netinet/in.h>** header shall define the **sockaddr_in6** structure, which shall include at least the following members:

10386	sa_family_t	sin6_family	AF_INET6.
10387	in_port_t	sin6_port	Port number.
10388	uint32_t	sin6_flowinfo	IPv6 traffic class and flow information.
10389	struct in6_addr	sin6_addr	IPv6 address.
10390	uint32_t	sin6_scope_id	Set of interfaces for a scope.

10391 The *sin6_port* and *sin6_addr* members shall be in network byte order.

10392 Prior to calling a function in this standard which reads values from a **sockaddr_in6** structure (for example, *bind()* or *connect()*), the application shall ensure that all members of the structure, including any additional non-standard members, if any, are initialized. If the **sockaddr_in6** structure has a non-standard member, and that member has a value other than the value that would result from default initialization, the behavior of any function in this standard that reads values from the **sockaddr_in6** structure is implementation-defined. All functions in this standard that return data in a **sockaddr_in6** structure (for example, *getaddrinfo()* or *accept()*) shall initialize the structure in a way that meets the above requirements, and shall ensure that

10400 each non-standard member, if any, has a value that produces the same behavior as default
 10401 initialization would in all functions in this standard which read values from a **sockaddr_in6**
 10402 structure.

10403 The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the
 10404 scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, the application
 10405 shall ensure that *sin6_scope_id* is a link index. For a site scope *sin6_addr*, the application shall
 10406 ensure that *sin6_scope_id* is a site index. The mapping of *sin6_scope_id* to an interface or set of
 10407 interfaces is implementation-defined.

10408 The <netinet/in.h> header shall declare the following external variable:

```
10409 const struct in6_addr in6addr_any
```

10410 This variable is initialized by the system to contain the wildcard IPv6 address. The
 10411 <netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be
 10412 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 10413 IPv6 wildcard address.

10414 The <netinet/in.h> header shall declare the following external variable:

```
10415 const struct in6_addr in6addr_loopback
```

10416 This variable is initialized by the system to contain the loopback IPv6 address. The
 10417 <netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
 10418 constant at compile time and can be used to initialize a variable of type **struct in6_addr** to the
 10419 IPv6 loopback address.

10420 The <netinet/in.h> header shall define the **ipv6_mreq** structure, which shall include at least the
 10421 following members:

```
10422 struct in6_addr  ipv6mr_multiaddr  IPv6 multicast address.  

10423 unsigned        ipv6mr_interface  Interface index.
```

10424 The <netinet/in.h> header shall define the following symbolic constants for use as values of the
 10425 *level* argument of *getsockopt()* and *setsockopt()*:

10426		IPPROTO_IP	Internet protocol.
10427	IP6	IPPROTO_IPV6	Internet Protocol Version 6.
10428		IPPROTO_ICMP	Control message protocol.
10429	RS	IPPROTO_RAW	Raw IP Packets Protocol.
10430		IPPROTO_TCP	Transmission control protocol.
10431		IPPROTO_UDP	User datagram protocol.

10432 The <netinet/in.h> header shall define the following symbolic constant for use as a local address
 10433 in the structure passed to *bind()*:

```
10434 INADDR_ANY      IPv4 wildcard address.
```

10435 The <netinet/in.h> header shall define the following symbolic constant for use as a destination
 10436 address in the structures passed to *connect()*, *sendmsg()*, and *sendto()*:

```
10437 INADDR_BROADCAST IPv4 broadcast address.
```

10438 The <netinet/in.h> header shall define the following symbolic constant, with the value
 10439 specified, to help applications declare buffers of the proper size to store IPv4 addresses in string

10440 form:

10441 INET_ADDRSTRLEN 16. Length of the string form for IP.

10442 The *htonl()*, *htons()*, *ntohl()*, and *ntohs()* functions shall be available as described in
10443 <arpa/inet.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from
10444 <arpa/inet.h>.

10445 IP6 The <netinet/in.h> header shall define the following symbolic constant, with the value
10446 specified, to help applications declare buffers of the proper size to store IPv6 addresses in string
10447 form:

10448 INET6_ADDRSTRLEN 46. Length of the string form for IPv6.

10449 IP6 The <netinet/in.h> header shall define the following symbolic constants, with distinct integer
10450 values, for use in the *option_name* argument in the *getsockopt()* or *setsockopt()* functions at
10451 protocol level IPPROTO_IPV6:

10452 IPV6_JOIN_GROUP Join a multicast group.

10453 IPV6_LEAVE_GROUP Quit a multicast group.

10454 IPV6_MULTICAST_HOPS
10455 Multicast hop limit.

10456 IPV6_MULTICAST_IF Interface to use for outgoing multicast packets.

10457 IPV6_MULTICAST_LOOP
10458 Multicast packets are delivered back to the local application.

10459 IPV6_UNICAST_HOPS Unicast hop limit.

10460 IPV6_V6ONLY Restrict AF_INET6 socket to IPv6 communications only.

10461 The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.
10462 Each macro is of type **int** and takes a single argument of type **const struct in6_addr ***:

10463 IN6_IS_ADDR_UNSPECIFIED
10464 Unspecified address.

10465 IN6_IS_ADDR_LOOPBACK
10466 Loopback address.

10467 IN6_IS_ADDR_MULTICAST
10468 Multicast address.

10469 IN6_IS_ADDR_LINKLOCAL
10470 Unicast link-local address.

10471 IN6_IS_ADDR_SITELOCAL
10472 Unicast site-local address.

10473 IN6_IS_ADDR_V4MAPPED
10474 IPv4 mapped address.

10475 IN6_IS_ADDR_V4COMPAT
10476 IPv4-compatible address.

10477 IN6_IS_ADDR_MC_NODELOCAL
10478 Multicast node-local address.

```

10479     IN6_IS_ADDR_MC_LINKLOCAL
10480         Multicast link-local address.
10481     IN6_IS_ADDR_MC_SITELOCAL
10482         Multicast site-local address.
10483     IN6_IS_ADDR_MC_ORGLOCAL
10484         Multicast organization-local address.
10485     IN6_IS_ADDR_MC_GLOBAL
10486         Multicast global address.
    
```

10487 **APPLICATION USAGE**

10488 Although applications are required to initialize all members (including any non-standard ones)
 10489 of a **sockaddr_in6** structure, the same is not required for the **sockaddr_in** structure, since
 10490 historically many applications only initialized the standard members. Despite this, applications
 10491 are encouraged to initialize **sockaddr_in** structures in a manner similar to the required
 10492 initialization of **sockaddr_in6** structures.

10493 Although it is common practice to initialize a **sockaddr_in6** structure using:

```

10494     struct sockaddr_in6 sa;
10495     memset(&sa, 0, sizeof sa);
    
```

10496 this method is not portable according to this standard, because the structure can contain pointer
 10497 or floating-point members that are not required to have an all-bits-zero representation after
 10498 default initialization. Portable methods make use of default initialization; for example:

```

10499     struct sockaddr_in6 sa = { 0 };
    
```

10500 or:

```

10501     static struct sockaddr_in6 sa_init;
10502     struct sockaddr_in6 sa = sa_init;
    
```

10503 A future version of this standard may require that a pointer object with an all-bits-zero
 10504 representation is a null pointer, and that **sockaddr_in6** does not have any floating-point
 10505 members if a floating-point object with an all-bits-zero representation does not have the value
 10506 0.0.

10507 **RATIONALE**

10508 The INADDR_ANY and INADDR_BROADCAST values are byte-order-neutral and thus their
 10509 byte order is not specified. Many implementations have additional constants as extensions, such
 10510 as INADDR_LOOPBACK, that are not byte-order-neutral. Traditionally, these constants are in
 10511 host byte order, requiring the use of *htonl()* when using them in a **sockaddr_in** structure.

10512 **FUTURE DIRECTIONS**

10513 None.

10514 **SEE ALSO**

10515 [Section 4.10](#) (on page 110), [<arpa/inet.h>](#), [<inttypes.h>](#), [<sys/socket.h>](#)

10516 [XSH connect\(\)](#), [getsockopt\(\)](#), [htonl\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#)

10517 **CHANGE HISTORY**

10518 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10519 The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base
 10520 Resolution bwg2001-004.

10521 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to
10522 the *in6addr_any* and *in6addr_loopback* external variables.

10523 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which
10524 structure members are in network byte order.

10525 **Issue 7**

10526 This reference page is clarified with respect to macros and symbolic constants.

10527 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0061 [355] is applied.

10528 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0065 [934], XBD/TC2-2008/0066 [952],
10529 XBD/TC2-2008/0067 [934], and XBD/TC2-2008/0068 [952] are applied.

10530 **NAME**

10531 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10532 **SYNOPSIS**

10533 #include <netinet/tcp.h>

10534 **DESCRIPTION**

10535 The <netinet/tcp.h> header shall define the following symbolic constant for use as a socket
10536 option at the IPPROTO_TCP level:

10537 TCP_NODELAY Avoid coalescing of small segments.

10538 The implementation need not allow the value of the option to be set via *setsockopt()* or retrieved
10539 via *getsockopt()*.

10540 **APPLICATION USAGE**

10541 None.

10542 **RATIONALE**

10543 None.

10544 **FUTURE DIRECTIONS**

10545 None.

10546 **SEE ALSO**

10547 <sys/socket.h>

10548 XSH *getsockopt()*, *setsockopt()*

10549 **CHANGE HISTORY**

10550 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10551 **Issue 7**

10552 This reference page is clarified with respect to macros and symbolic constants.

10553 **NAME**10554 `nl_types.h` ‡'data types10555 **SYNOPSIS**10556 `#include <nl_types.h>`10557 **DESCRIPTION**10558 The **<nl_types.h>** header shall define at least the following types:10559 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*
10560 to identify a catalog descriptor.10561 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects
10562 of type **nl_item** are defined in **<langinfo.h>**.10563 The **<nl_types.h>** header shall define at least the following symbolic constants:10564 **NL_SETD** Used by *gencat* when no *\$set* directive is specified in a message text source
10565 file. This constant can be passed as the value of *set_id* on subsequent calls
10566 to *catgets()* (that is, to retrieve messages from the default message set).
10567 The value of **NL_SETD** is implementation-defined.10568 **NL_CAT_LOCALE** Value that must be passed as the *oflag* argument to *catopen()* to ensure that
10569 message catalog selection depends on the *LC_MESSAGES* locale category,
10570 rather than directly on the *LANG* environment variable.10571 The following shall be declared as functions and may also be defined as macros. Function
10572 prototypes shall be provided.10573 `int catclose(nl_catd);`
10574 `char *catgets(nl_catd, int, int, const char *);`
10575 `nl_catd catopen(const char *, int);`10576 **APPLICATION USAGE**

10577 None.

10578 **RATIONALE**

10579 None.

10580 **FUTURE DIRECTIONS**

10581 None.

10582 **SEE ALSO**10583 [<langinfo.h>](#)10584 XSH *catclose()*, *catgets()*, *catopen()*, *nl_langinfo()*10585 XCU *gencat*10586 **CHANGE HISTORY**

10587 First released in Issue 2.

10588 **Issue 7**10589 The **<nl_types.h>** header is moved from the XSI option to the Base.

10590 This reference page is clarified with respect to macros and symbolic constants.

10591 **NAME**

10592 poll.h ¶definitions for the poll() function

10593 **SYNOPSIS**

10594 #include <poll.h>

10595 **DESCRIPTION**

10596 The <poll.h> header shall define the **pollfd** structure, which shall include at least the following
10597 members:

10598 int fd The following descriptor being polled.
10599 short events The input event flags (see below).
10600 short revents The output event flags (see below).

10601 The <poll.h> header shall define the following type through **typedef**:

10602 **nfds_t** An unsigned integer type used for the number of file descriptors.

10603 The implementation shall support one or more programming environments in which the width
10604 of **nfds_t** is no greater than the width of type **long**. The names of these programming
10605 environments can be obtained using the *confstr()* function or the *getconf* utility.

10606 The <poll.h> header shall define the following symbolic constants, zero or more of which may
10607 be OR'ed together to form the *events* or *revents* members in the **pollfd** structure:

10608 POLLIN Data other than high-priority data may be read without blocking.
10609 POLLRDNORM Normal data may be read without blocking.
10610 POLLRDBAND Priority data may be read without blocking.
10611 POLLPRI High priority data may be read without blocking.
10612 POLLOUT Normal data may be written without blocking.
10613 POLLWRNORM Equivalent to POLLOUT.
10614 POLLWRBAND Priority data may be written.
10615 POLLERR An error has occurred (*revents* only).
10616 POLLHUP Device has been disconnected (*revents* only).
10617 POLLNVAL Invalid *fd* member (*revents* only).

10618 The significance and semantics of normal, priority, and high-priority data are file and device-
10619 specific.

10620 The following shall be declared as a function and may also be defined as a macro. A function
10621 prototype shall be provided.

10622 int poll(struct pollfd [], nfds_t, int);

10623 **APPLICATION USAGE**

10624 None.

10625 **RATIONALE**

10626 None.

10627 **FUTURE DIRECTIONS**

10628 None.

10629 **SEE ALSO**10630 XSH *confstr()*, *poll()*10631 XCU *getconf*10632 **CHANGE HISTORY**

10633 First released in Issue 4, Version 2.

10634 **Issue 6**10635 The description of the symbolic constants is updated to match the *poll()* function.10636 Text related to STREAMS has been moved to the *poll()* reference page.10637 A note is added to the DESCRIPTION regarding the significance and semantics of normal,
10638 priority, and high-priority data.10639 **Issue 7**10640 The **<poll.h>** header is moved from the XSI option to the Base.

10641 **NAME**

10642 pthread.h — threads

10643 **SYNOPSIS**

10644 #include <pthread.h>

10645 **DESCRIPTION**

10646 The <pthread.h> header shall define the following symbolic constants:

- 10647 PTHREAD_BARRIER_SERIAL_THREAD
- 10648 PTHREAD_CANCEL_ASYNCHRONOUS
- 10649 PTHREAD_CANCEL_ENABLE
- 10650 PTHREAD_CANCEL_DEFERRED
- 10651 PTHREAD_CANCEL_DISABLE
- 10652 PTHREAD_CANCELED
- 10653 PTHREAD_CREATE_DETACHED
- 10654 PTHREAD_CREATE_JOINABLE
- 10655 TPS PTHREAD_EXPLICIT_SCHED
- 10656 PTHREAD_INHERIT_SCHED
- 10657 PTHREAD_MUTEX_DEFAULT
- 10658 PTHREAD_MUTEX_ERRORCHECK
- 10659 PTHREAD_MUTEX_NORMAL
- 10660 PTHREAD_MUTEX_RECURSIVE
- 10661 PTHREAD_MUTEX_ROBUST
- 10662 PTHREAD_MUTEX_STALLED
- 10663 PTHREAD_ONCE_INIT
- 10664 RPI | TPI PTHREAD_PRIO_INHERIT
- 10665 MC1 PTHREAD_PRIO_NONE
- 10666 RPP | TPP PTHREAD_PRIO_PROTECT
- 10667 PTHREAD_PROCESS_SHARED
- 10668 PTHREAD_PROCESS_PRIVATE
- 10669 TPS PTHREAD_SCOPE_PROCESS
- 10670 PTHREAD_SCOPE_SYSTEM

10671 The <pthread.h> header shall define the following compile-time constant expressions valid as
 10672 initializers for the following types:

Name	Initializer for Type
PTHREAD_COND_INITIALIZER	pthread_cond_t
PTHREAD_MUTEX_INITIALIZER	pthread_mutex_t
PTHREAD_RWLOCK_INITIALIZER	pthread_rwlock_t

10677 The <pthread.h> header shall define the pthread_attr_t, pthread_barrier_t,
 10678 pthread_barrierattr_t, pthread_cond_t, pthread_condattr_t, pthread_key_t, pthread_mutex_t,
 10679 pthread_mutexattr_t, pthread_once_t, pthread_rwlock_t, pthread_rwlockattr_t,
 10680 pthread_spinlock_t, and pthread_t types as described in <sys/types.h>.

10681 The following shall be declared as functions and may also be defined as macros. Function
 10682 prototypes shall be provided.

```

10683 int pthread_atfork(void (*)(void), void (*)(void),
10684                  void (*)(void));
10685 int pthread_attr_destroy(pthread_attr_t *);
10686 int pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10687 int pthread_attr_getguardsize(const pthread_attr_t *restrict,
```

```

10688         size_t *restrict);
10689 TPS    int  pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10690         int *restrict);
10691        int  pthread_attr_getschedparam(const pthread_attr_t *restrict,
10692         struct sched_param *restrict);
10693 TPS    int  pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10694         int *restrict);
10695        int  pthread_attr_getscope(const pthread_attr_t *restrict,
10696         int *restrict);
10697 TSA TSS int  pthread_attr_getstack(const pthread_attr_t *restrict,
10698         void **restrict, size_t *restrict);
10699 TSS    int  pthread_attr_getstacksize(const pthread_attr_t *restrict,
10700         size_t *restrict);
10701        int  pthread_attr_init(pthread_attr_t *);
10702        int  pthread_attr_setdetachstate(pthread_attr_t *, int);
10703        int  pthread_attr_setguardsize(pthread_attr_t *, size_t);
10704 TPS    int  pthread_attr_setinheritsched(pthread_attr_t *, int);
10705        int  pthread_attr_setschedparam(pthread_attr_t *restrict,
10706         const struct sched_param *restrict);
10707 TPS    int  pthread_attr_setschedpolicy(pthread_attr_t *, int);
10708        int  pthread_attr_setscope(pthread_attr_t *, int);
10709 TSA TSS int  pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10710 TSS    int  pthread_attr_setstacksize(pthread_attr_t *, size_t);
10711        int  pthread_barrier_destroy(pthread_barrier_t *);
10712        int  pthread_barrier_init(pthread_barrier_t *restrict,
10713         const pthread_barrierattr_t *restrict, unsigned);
10714        int  pthread_barrier_wait(pthread_barrier_t *);
10715        int  pthread_barrierattr_destroy(pthread_barrierattr_t *);
10716 TSH    int  pthread_barrierattr_getpshared(
10717         const pthread_barrierattr_t *restrict, int *restrict);
10718        int  pthread_barrierattr_init(pthread_barrierattr_t *);
10719 TSH    int  pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10720        int  pthread_cancel(pthread_t);
10721        int  pthread_cond_broadcast(pthread_cond_t *);
10722        int  pthread_cond_destroy(pthread_cond_t *);
10723        int  pthread_cond_init(pthread_cond_t *restrict,
10724         const pthread_condattr_t *restrict);
10725        int  pthread_cond_signal(pthread_cond_t *);
10726        int  pthread_cond_timedwait(pthread_cond_t *restrict,
10727         pthread_mutex_t *restrict, const struct timespec *restrict);
10728        int  pthread_cond_wait(pthread_cond_t *restrict,
10729         pthread_mutex_t *restrict);
10730        int  pthread_condattr_destroy(pthread_condattr_t *);
10731        int  pthread_condattr_getclock(const pthread_condattr_t *restrict,
10732         clockid_t *restrict);
10733 TSH    int  pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10734         int *restrict);
10735        int  pthread_condattr_init(pthread_condattr_t *);
10736        int  pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10737 TSH    int  pthread_condattr_setpshared(pthread_condattr_t *, int);
10738        int  pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10739         void *(*)(void*), void *restrict);

```

```

10740     int   pthread_detach(pthread_t);
10741     int   pthread_equal(pthread_t, pthread_t);
10742     void  pthread_exit(void *);
10743     OB XSI int   pthread_getconcurrency(void);
10744     TCT   int   pthread_getcpuclockid(pthread_t, clockid_t *);
10745     TPS   int   pthread_getschedparam(pthread_t, int *restrict,
10746         struct sched_param *restrict);
10747     void *pthread_getspecific(pthread_key_t);
10748     int   pthread_join(pthread_t, void **);
10749     int   pthread_key_create(pthread_key_t *, void (*)(void*));
10750     int   pthread_key_delete(pthread_key_t);
10751     int   pthread_mutex_consistent(pthread_mutex_t *);
10752     int   pthread_mutex_destroy(pthread_mutex_t *);
10753     RPP|TPP int   pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10754         int *restrict);
10755     int   pthread_mutex_init(pthread_mutex_t *restrict,
10756         const pthread_mutexattr_t *restrict);
10757     int   pthread_mutex_lock(pthread_mutex_t *);
10758     RPP|TPP int   pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10759         int *restrict);
10760     int   pthread_mutex_timedlock(pthread_mutex_t *restrict,
10761         const struct timespec *restrict);
10762     int   pthread_mutex_trylock(pthread_mutex_t *);
10763     int   pthread_mutex_unlock(pthread_mutex_t *);
10764     int   pthread_mutexattr_destroy(pthread_mutexattr_t *);
10765     RPP|TPP int   pthread_mutexattr_getprioceiling(
10766         const pthread_mutexattr_t *restrict, int *restrict);
10767     MC1   int   pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10768         int *restrict);
10769     TSH   int   pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10770         int *restrict);
10771     int   pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict,
10772         int *restrict);
10773     int   pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10774         int *restrict);
10775     int   pthread_mutexattr_init(pthread_mutexattr_t *);
10776     RPP|TPP int   pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10777     MC1   int   pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10778     TSH   int   pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10779     int   pthread_mutexattr_setrobust(pthread_mutexattr_t *, int);
10780     int   pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10781     int   pthread_once(pthread_once_t *, void (*)(void));
10782     int   pthread_rwlock_destroy(pthread_rwlock_t *);
10783     int   pthread_rwlock_init(pthread_rwlock_t *restrict,
10784         const pthread_rwlockattr_t *restrict);
10785     int   pthread_rwlock_rdlock(pthread_rwlock_t *);
10786     int   pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10787         const struct timespec *restrict);
10788     int   pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10789         const struct timespec *restrict);
10790     int   pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10791     int   pthread_rwlock_trywrlock(pthread_rwlock_t *);

```

```

10792     int  pthread_rwlock_unlock(pthread_rwlock_t *);
10793     int  pthread_rwlock_wrlock(pthread_rwlock_t *);
10794     int  pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10795 TSH   int  pthread_rwlockattr_getpshared(
10796         const pthread_rwlockattr_t *restrict, int *restrict);
10797     int  pthread_rwlockattr_init(pthread_rwlockattr_t *);
10798 TSH   int  pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10799     pthread_t
10800         pthread_self(void);
10801     int  pthread_setcancelstate(int, int *);
10802     int  pthread_setcanceltype(int, int *);
10803 OB XSI int  pthread_setconcurrency(int);
10804 TPS   int  pthread_setschedparam(pthread_t, int,
10805         const struct sched_param *);
10806     int  pthread_setschedprio(pthread_t, int);
10807     int  pthread_setspecific(pthread_key_t, const void *);
10808     int  pthread_spin_destroy(pthread_spinlock_t *);
10809     int  pthread_spin_init(pthread_spinlock_t *, int);
10810     int  pthread_spin_lock(pthread_spinlock_t *);
10811     int  pthread_spin_trylock(pthread_spinlock_t *);
10812     int  pthread_spin_unlock(pthread_spinlock_t *);
10813     void pthread_testcancel(void);

```

10814 The following may be declared as functions, or defined as macros, or both. If functions are
 10815 declared, function prototypes shall be provided.

```

10816         pthread_cleanup_pop()
10817         pthread_cleanup_push()

```

10818 Inclusion of the **<pthread.h>** header shall make symbols defined in the headers **<sched.h>** and
 10819 **<time.h>** visible.

10820 APPLICATION USAGE

10821 None.

10822 RATIONALE

10823 None.

10824 FUTURE DIRECTIONS

10825 None.

10826 SEE ALSO

10827 **<sched.h>**, **<sys/types.h>**, **<time.h>**

```

10828 XSH pthread_atfork(), pthread_attr_destroy(), pthread_attr_getdetachstate(),
10829 pthread_attr_getguardsize(), pthread_attr_getinheritsched(), pthread_attr_getschedparam(),
10830 pthread_attr_getschedpolicy(), pthread_attr_getscope(), pthread_attr_getstack(),
10831 pthread_attr_getstacksize(), pthread_barrier_destroy(), pthread_barrier_wait(),
10832 pthread_barrierattr_destroy(), pthread_barrierattr_getpshared(), pthread_cancel(),
10833 pthread_cleanup_pop(), pthread_cond_broadcast(), pthread_cond_destroy(), pthread_cond_timedwait(),
10834 pthread_condattr_destroy(), pthread_condattr_getclock(), pthread_condattr_getpshared(),
10835 pthread_create(), pthread_detach(), pthread_equal(), pthread_exit(), pthread_getconcurrency(),
10836 pthread_getcpuclockid(), pthread_getschedparam(), pthread_getspecific(), pthread_join(),
10837 pthread_key_create(), pthread_key_delete(), pthread_mutex_consistent(), pthread_mutex_destroy(),
10838 pthread_mutex_getprioceiling(), pthread_mutex_lock(), pthread_mutex_timedlock(),

```

10839 *pthread_mutexattr_destroy()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
 10840 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_getrobust()*, *pthread_mutexattr_gettype()*,
 10841 *pthread_once()*, *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
 10842 *pthread_rwlock_timedwrlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*,
 10843 *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_self()*,
 10844 *pthread_setcancelstate()*, *pthread_setschedprio()*, *pthread_spin_destroy()*, *pthread_spin_lock()*,
 10845 *pthread_spin_unlock()*

10846 CHANGE HISTORY

10847 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

10848 Issue 6

10849 The RTT margin markers are broken out into their POSIX options.

10850 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the
 10851 *pthread_cond_wait()* function.

10852 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
 10853 *pthread_setschedparam()* function so that its second argument is of type **int**.

10854 The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment
 10855 with IEEE Std 1003.1d-1999.

10856 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10857 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*,
 10858 *pthread_barrierattr_destroy()*, *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*,
 10859 *pthread_barrierattr_setpshared()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*,
 10860 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_spin_destroy()*,
 10861 *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, and *pthread_spin_unlock()*.

10862 PTHREAD_RWLOCK_INITIALIZER is removed for alignment with IEEE Std 1003.1j-2000.

10863 Functions previously marked as part of the Read-Write Locks option are now moved to the
 10864 Threads option.

10865 The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize()*,
 10866 *pthread_attr_getinheritsched()*, *pthread_attr_getschedparam()*, *pthread_attr_getschedpolicy()*,
 10867 *pthread_attr_getscope()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*,
 10868 *pthread_attr_setschedparam()*, *pthread_barrier_init()*, *pthread_barrierattr_getpshared()*,
 10869 *pthread_cond_init()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, *pthread_cond_wait()*,
 10870 *pthread_condattr_getclock()*, *pthread_condattr_getpshared()*, *pthread_create()*,
 10871 *pthread_getschedparam()*, *pthread_mutex_getprioceiling()*, *pthread_mutex_init()*,
 10872 *pthread_mutex_setprioceiling()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
 10873 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_gettype()*, *pthread_rwlock_init()*,
 10874 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlockattr_getpshared()*, and
 10875 *pthread_sigmask()*.

10876 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and
 10877 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI option.

10878 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for
 10879 the *pthread_kill()* and *pthread_sigmask()* functions. These are required to be in the <signal.h>
 10880 header. They are allowed here through the name space rules.

10881 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio()* function.

10882 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors
 10883 that were in contradiction with the System Interfaces volume of POSIX.1-2017.

10884 **Issue 7**

10885 SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread_mutex_timedlock()*
10886 function prototype.

10887 SD5-XBD-ERN-62 is applied.

10888 Austin Group Interpretation 1003.1-2001 #048 is applied, reinstating the
10889 PTHREAD_RWLOCK_INITIALIZER symbol.

10890 The **<pthread.h>** header is moved from the Threads option to the Base.

10891 The following extended mutex types are moved from the XSI option to the Base:

10892 PTHREAD_MUTEX_NORMAL
10893 PTHREAD_MUTEX_ERRORCHECK
10894 PTHREAD_MUTEX_RECURSIVE
10895 PTHREAD_MUTEX_DEFAULT

10896 The PTHREAD_MUTEX_ROBUST and PTHREAD_MUTEX_STALLED symbols and the
10897 *pthread_mutex_consistent()*, *pthread_mutexattr_getrobust()*, and *pthread_mutexattr_setrobust()*
10898 functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

10899 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options
10900 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex
10901 or Robust Mutex Priority Inheritance, respectively.

10902 This reference page is clarified with respect to macros and symbolic constants.

10903 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0069 [624] is applied.

10904 **NAME**

10905 pwd.h †'password structure

10906 **SYNOPSIS**

10907 #include <pwd.h>

10908 **DESCRIPTION**

10909 The <pwd.h> header shall define the **struct passwd**, structure, which shall include at least the
10910 following members:

10911	char	*pw_name	User's login name.
10912	uid_t	pw_uid	Numerical user ID.
10913	gid_t	pw_gid	Numerical group ID.
10914	char	*pw_dir	Initial working directory.
10915	char	*pw_shell	Program to use as shell.

10916 The <pwd.h> header shall define the **gid_t**, **uid_t**, and **size_t** types as described in
10917 <sys/types.h>.

10918 The following shall be declared as functions and may also be defined as macros. Function
10919 prototypes shall be provided.

```

10920 XSI void      endpwent(void);
10921 struct passwd *getpwent(void);
10922 struct passwd *getpwnam(const char *);
10923 int      getpwnam_r(const char *, struct passwd *, char *,
10924                  size_t, struct passwd **);
10925 struct passwd *getpwuid(uid_t);
10926 int      getpwuid_r(uid_t, struct passwd *, char *,
10927                  size_t, struct passwd **);
10928 XSI void      setpwent(void);

```

10929 **APPLICATION USAGE**

10930 None.

10931 **RATIONALE**

10932 None.

10933 **FUTURE DIRECTIONS**

10934 None.

10935 **SEE ALSO**

10936 <sys/types.h>

10937 XSH *endpwent()*, *getpwnam()*, *getpwuid()*

10938 **CHANGE HISTORY**

10939 First released in Issue 1.

10940 **Issue 5**

10941 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10942 **Issue 6**

10943 The following new requirements on POSIX implementations derive from alignment with the
10944 Single UNIX Specification:

10945 The **gid_t** and **uid_t** types are mandated.

10946 The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the Thread-Safe
10947 Functions option.

10948 **Issue 7**

10949 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

10950 **NAME**

10951 regex.h — regular expression matching types

10952 **SYNOPSIS**

10953 #include <regex.h>

10954 **DESCRIPTION**

10955 The <regex.h> header shall define the structures and symbolic constants used by the *regcomp()*,
10956 *regex()*, *regerror()*, and *regfree()* functions.

10957 The <regex.h> header shall define the **regex_t** structure type, which shall include at least the
10958 following member:

10959 size_t re_nsub Number of parenthesized subexpressions.

10960 The <regex.h> header shall define the **size_t** type as described in <sys/types.h>.

10961 The <regex.h> header shall define the **regoff_t** type as a signed integer type that can hold the
10962 largest value that can be stored in either a **ptrdiff_t** type or a **ssize_t** type.

10963 The <regex.h> header shall define the **regmatch_t** structure type, which shall include at least the
10964 following members:

10965	regoff_t	rm_so	Byte offset from start of string
10966			to start of substring.
10967	regoff_t	rm_eo	Byte offset from start of string of the
10968			first character after the end of substring.

10969 The <regex.h> header shall define the following symbolic constants for the *cflags* parameter to
10970 the *regcomp()* function:

10971	REG_EXTENDED	Use Extended Regular Expressions.
10972	REG_ICASE	Ignore case in match.
10973	REG_NOSUB	Report only success or fail in <i>regex()</i> .
10974	REG_NEWLINE	Change the handling of <newline>.

10975 The <regex.h> header shall define the following symbolic constants for the *eflags* parameter to
10976 the *regex()* function:

10977	REG_NOTBOL	The <circumflex> character ('^'), when taken as a special character, does
10978		not match the beginning of <i>string</i> .
10979	REG_NOTEOL	The <dollar-sign> ('\$'), when taken as a special character, does not
10980		match the end of <i>string</i> .

10981 The <regex.h> header shall define the following symbolic constants as error return values:

10982	REG_NOMATCH	<i>regex()</i> failed to match.
10983	REG_BADPAT	Invalid regular expression.
10984	REG_ECOLLATE	Invalid collating element referenced.
10985	REG_ETYPE	Invalid character class type referenced.
10986	REG_ESCAPE	Trailing <backslash> character in pattern.
10987	REG_ESUBREG	Number in <i>\digit</i> invalid or in error.

10988 REG_EBRACK "[]" imbalance.

10989 REG_EPAREN "\\(\\)" or "\\(" imbalance.

10990 REG_EBRACE "\\{\\}" imbalance.

10991 REG_BADBR Content of "\\{\\}" invalid: not a number, number too large, more than
10992 two numbers, first larger than second.

10993 REG_ERANGE Invalid endpoint in range expression.

10994 REG_ESPACE Out of memory.

10995 REG_BADRPT '?', '*', or '+' not preceded by valid regular expression.

10996 The following shall be declared as functions and may also be defined as macros. Function
10997 prototypes shall be provided.

```
10998 int regcomp(regex_t *restrict, const char *restrict, int);
10999 size_t regerror(int, const regex_t *restrict, char *restrict, size_t);
11000 int regexec(const regex_t *restrict, const char *restrict, size_t,
11001 regmatch_t [restrict], int);
11002 void regfree(regex_t *);
```

11003 The implementation may define additional macros or constants using names beginning with
11004 REG_.

11005 APPLICATION USAGE

11006 None.

11007 RATIONALE

11008 None.

11009 FUTURE DIRECTIONS

11010 None.

11011 SEE ALSO11012 [<sys/types.h>](#)11013 XSH *regcomp()***11014 CHANGE HISTORY**

11015 First released in Issue 4.

11016 Originally derived from the ISO POSIX-2 standard.

11017 Issue 6

11018 The REG_ENOSYS constant is marked obsolescent.

11019 The **restrict** keyword is added to the prototypes for *regcomp()*, *regerror()*, and *regexec()*.11020 A statement is added that the **size_t** type is defined as described in [<sys/types.h>](#).**11021 Issue 7**

11022 SD5-XBD-ERN-60 is applied.

11023 The obsolescent REG_ENOSYS constant is removed.

11024 This reference page is clarified with respect to macros and symbolic constants.

11025 **NAME**

11026 sched.h ‡execution scheduling

11027 **SYNOPSIS**

11028 #include <sched.h>

11029 **DESCRIPTION**

11030 PS The <sched.h> header shall define the **pid_t** type as described in <sys/types.h>.

11031 SS|TSP The <sched.h> header shall define the **time_t** type as described in <sys/types.h>.

11032 The <sched.h> header shall define the **timespec** structure as described in <time.h>.

11033 The <sched.h> header shall define the **sched_param** structure, which shall include the
 11034 scheduling parameters required for implementation of each supported scheduling policy. This
 11035 structure shall include at least the following member:

11036 int sched_priority Process or thread execution scheduling priority.

11037 SS|TSP The **sched_param** structure defined in <sched.h> shall include the following members in
 11038 addition to those specified above:

11039 int sched_ss_low_priority Low scheduling priority for
 11040 sporadic server.

11041 struct timespec sched_ss_repl_period Replenishment period for
 11042 sporadic server.

11043 struct timespec sched_ss_init_budget Initial budget for sporadic server.

11044 int sched_ss_max_repl Maximum pending replenishments for
 11045 sporadic server.

11046 Each process or thread is controlled by an associated scheduling policy and priority. Associated
 11047 with each policy is a priority range. Each policy definition specifies the minimum priority range
 11048 for that policy. The priority ranges for each policy may overlap the priority ranges of other
 11049 policies.

11050 Four scheduling policies are defined; others may be defined by the implementation. The four
 11051 standard policies are indicated by the values of the following symbolic constants:

11052 PS|TPS **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

11053 PS|TPS **SCHED_RR** Round robin scheduling policy.

11054 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

11055 PS|TPS **SCHED_OTHER** Another scheduling policy.

11056 The values of these constants are distinct.

11057 The following shall be declared as functions and may also be defined as macros. Function
 11058 prototypes shall be provided.

11059 PS|TPS int sched_get_priority_max(int);

11060 int sched_get_priority_min(int);

11061 PS int sched_getparam(pid_t, struct sched_param *);

11062 int sched_getscheduler(pid_t);

11063 PS|TPS int sched_rr_get_interval(pid_t, struct timespec *);

11064 PS int sched_setparam(pid_t, const struct sched_param *);

11065 int sched_setscheduler(pid_t, int, const struct sched_param *);

11066 int sched_yield(void);

- 11067 Inclusion of the **<sched.h>** header may make visible all symbols from the **<time.h>** header.
- 11068 **APPLICATION USAGE**
- 11069 None.
- 11070 **RATIONALE**
- 11071 None.
- 11072 **FUTURE DIRECTIONS**
- 11073 None.
- 11074 **SEE ALSO**
- 11075 [<sys/types.h>](#), [<time.h>](#)
- 11076 XSH *sched_get_priority_max()*, *sched_getparam()*, *sched_getscheduler()*, *sched_rr_get_interval()*,
11077 *sched_setparam()*, *sched_setscheduler()*, *sched_yield()*
- 11078 **CHANGE HISTORY**
- 11079 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 11080 **Issue 6**
- 11081 The **<sched.h>** header is marked as part of the Process Scheduling option.
- 11082 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
11083 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.
- 11084 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
11085 members *sched_ss_repl_period* and *sched_ss_init_budget* should be type **struct timespec** and not
11086 **timespec**.
- 11087 Symbols from **<time.h>** may be made visible when **<sched.h>** is included.
- 11088 IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied,
11089 aligning the function prototype shading and margin codes with the System Interfaces volume of
11090 IEEE Std 1003.1-2001.
- 11091 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the
11092 DESCRIPTION to differentiate between thread and process execution.
- 11093 **Issue 7**
- 11094 SD5-XBD-ERN-13 is applied.
- 11095 Austin Group Interpretation 1003.1-2001 #064 is applied, correcting the options markings.
- 11096 The **<sched.h>** headers is moved from the Threads option to the Base.
- 11097 Declarations for the **pid_t** and **time_t** types and the **timespec** structure are added.

11098 **NAME**

11099 search.h — search tables

11100 **SYNOPSIS**11101 XSI `#include <search.h>`11102 **DESCRIPTION**

11103 The **<search.h>** header shall define the **ENTRY** type for structure **entry** which shall include the
 11104 following members:

```
11105 char    *key
11106 void    *data
```

11107 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as
 11108 follows:

```
11109 enum { FIND, ENTER } ACTION;
11110 enum { preorder, postorder, endorder, leaf } VISIT;
```

11111 The **<search.h>** header shall define the **size_t** type as described in **<sys/types.h>**.

11112 The following shall be declared as functions and may also be defined as macros. Function
 11113 prototypes shall be provided.

```
11114 int    hcreate(size_t);
11115 void   hdestroy(void);
11116 ENTRY *hsearch(ENTRY, ACTION);
11117 void   insque(void *, void *);
11118 void   *lfind(const void *, const void *, size_t *,
11119             size_t, int (*)(const void *, const void *));
11120 void   *lsearch(const void *, void *, size_t *,
11121             size_t, int (*)(const void *, const void *));
11122 void   remque(void *);
11123 void   *tdelete(const void *restrict, void **restrict,
11124             int (*)(const void *, const void *));
11125 void   *tfind(const void *, void *const *,
11126             int (*)(const void *, const void *));
11127 void   *tsearch(const void *, void **,
11128             int (*)(const void *, const void *));
11129 void   twalk(const void *,
11130             void (*)(const void *, VISIT, int ));
```

11131 **APPLICATION USAGE**

11132 None.

11133 **RATIONALE**

11134 None.

11135 **FUTURE DIRECTIONS**

11136 None.

11137 **SEE ALSO**11138 [<sys/types.h>](#)11139 XSH [hcreate\(\)](#), [insque\(\)](#), [lsearch\(\)](#), [tdelete\(\)](#)

11140 **CHANGE HISTORY**

11141 First released in Issue 1. Derived from Issue 1 of the SVID.

11142 **Issue 6**

11143 The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and
11144 *tsearch()*.

11145 The **restrict** keyword is added to the prototype for *tdelete()*.

11146 **NAME**

11147 semaphore.h — semaphores

11148 **SYNOPSIS**

11149 #include <semaphore.h>

11150 **DESCRIPTION**

11151 The <semaphore.h> header shall define the **sem_t** type, used in performing semaphore
 11152 operations. The semaphore may be implemented using a file descriptor, in which case
 11153 applications are able to open up at least a total of {OPEN_MAX} files and semaphores.

11154 The <semaphore.h> header shall define the symbolic constant SEM_FAILED which shall have
 11155 type **sem_t** *.

11156 The following shall be declared as functions and may also be defined as macros. Function
 11157 prototypes shall be provided.

```

11158 int sem_close(sem_t *);
11159 int sem_destroy(sem_t *);
11160 int sem_getvalue(sem_t *restrict, int *restrict);
11161 int sem_init(sem_t *, int, unsigned);
11162 sem_t *sem_open(const char *, int, ...);
11163 int sem_post(sem_t *);
11164 int sem_timedwait(sem_t *restrict, const struct timespec *restrict);
11165 int sem_trywait(sem_t *);
11166 int sem_unlink(const char *);
11167 int sem_wait(sem_t *);

```

11168 Inclusion of the <semaphore.h> header may make visible symbols defined in the <fcntl.h> and
 11169 <time.h> headers.

11170 **APPLICATION USAGE**

11171 None.

11172 **RATIONALE**

11173 None.

11174 **FUTURE DIRECTIONS**

11175 None.

11176 **SEE ALSO**

11177 <fcntl.h>, <sys/types.h>, <time.h>

11178 XSH *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*,
 11179 *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*

11180 **CHANGE HISTORY**

11181 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

11182 **Issue 6**

11183 The <semaphore.h> header is marked as part of the Semaphores option.

11184 The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

11185 The *sem_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.11186 The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

11187 **Issue 7**

11188 SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>**
11189 header.

11190 The **<semaphore.h>** header is moved from the Semaphores option to the Base.

11191 This reference page is clarified with respect to macros and symbolic constants.

11192 **NAME**
 11193 setjmp.h — stack environment declarations

11194 **SYNOPSIS**
 11195 #include <setjmp.h>

11196 **DESCRIPTION**
 11197 CX Some of the functionality described on this reference page extends the ISO C standard.
 11198 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 11199 enable the visibility of these symbols in this header.

11200 CX The <setjmp.h> header shall define the array types **jmp_buf** and **sigjmp_buf**.
 11201 The following shall be declared as functions and may also be defined as macros. Function
 11202 prototypes shall be provided.

11203 OB XSI void _longjmp(jmp_buf, int);
 11204 void longjmp(jmp_buf, int);
 11205 CX void siglongjmp(sigjmp_buf, int);

11206 The following may be declared as functions, or defined as macros, or both. If functions are
 11207 declared, function prototypes shall be provided.

11208 OB XSI int _setjmp(jmp_buf);
 11209 int setjmp(jmp_buf);
 11210 CX int sigsetjmp(sigjmp_buf, int);

11211 **APPLICATION USAGE**
 11212 None.

11213 **RATIONALE**
 11214 None.

11215 **FUTURE DIRECTIONS**
 11216 None.

11217 **SEE ALSO**
 11218 XSH Section 2.2 (on page 472), *longjmp()*, *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*

11219 **CHANGE HISTORY**
 11220 First released in Issue 1.

11221 **Issue 6**
 11222 Extensions beyond the ISO C standard are marked.

11223 **Issue 7**
 11224 SD5-XBD-ERN-6 is applied.

11225 **NAME**

11226 signal.h ‡signals

11227 **SYNOPSIS**

11228 #include <signal.h>

11229 **DESCRIPTION**

11230 CX Some of the functionality described on this reference page extends the ISO C standard.
 11231 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 11232 enable the visibility of these symbols in this header.

11233 The **<signal.h>** header shall define the following macros, which shall expand to constant
 11234 expressions with distinct values that have a type compatible with the second argument to, and
 11235 the return value of, the *signal()* function, and whose values shall compare unequal to the
 11236 address of any declarable function.

11237 SIG_DFL Request for default signal handling.

11238 SIG_ERR Return value from *signal()* in case of error.

11239 OB XSI SIG_HOLD Request that signal be held.

11240 SIG_IGN Request that signal be ignored.

11241 CX The **<signal.h>** header shall define the **pthread_t**, **size_t**, and **uid_t** types as described in
 11242 **<sys/types.h>**.

11243 The **<signal.h>** header shall define the **timespec** structure as described in **<time.h>**.

11244 The **<signal.h>** header shall define the following data types:

11245 **sig_atomic_t** Possibly volatile-qualified integer type of an object that can be accessed as
 11246 an atomic entity, even in the presence of asynchronous interrupts.

11247 CX **sigset_t** Integer or structure type of an object used to represent sets of signals.

11248 CX **pid_t** As described in **<sys/types.h>**.

11249 CX The **<signal.h>** header shall define the **pthread_attr_t** type as described in **<sys/types.h>**.

11250 The **<signal.h>** header shall define the **sigevent** structure, which shall include at least the
 11251 following members:

11252	int	sigev_notify	Notification type.
11253	int	sigev_signo	Signal number.
11254	union sigval	sigev_value	Signal value.
11255	void	(*sigev_notify_function)(union sigval)	Notification function.
11256			Notification function.
11257	pthread_attr_t	*sigev_notify_attributes	Notification attributes.

11258 The **<signal.h>** header shall define the following symbolic constants for the values of
 11259 *sigev_notify*:

11260 SIGEV_NONE No asynchronous notification is delivered when the event of interest
 11261 occurs.

11262 SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when
 11263 the event of interest occurs.

11264 SIGEV_THREAD A notification function is called to perform notification.

11265 The **signal** union shall be defined as:

```
11266 int      sival_int      Integer signal value.  
11267 void    *sival_ptr     Pointer signal value.
```

11268 The <**signal.h**> header shall declare the SIGRTMIN and SIGRTMAX macros, which shall expand
11269 to positive integer expressions with type **int**, but which need not be constant expressions. These
11270 macros specify a range of signal numbers that are reserved for application use and for which the
11271 realtime signal behavior specified in this volume of POSIX.1-2017 is supported. The signal
11272 numbers in this range do not overlap any of the signals specified in the following table.

11273 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal
11274 numbers.

11275 It is implementation-defined whether realtime signal behavior is supported for other signals.

11276 The <**signal.h**> header shall define the following macros that are used to refer to the signals that
11277 occur in the system. Signals defined here begin with the letters SIG followed by an uppercase
11278 letter. The macros shall expand to positive integer constant expressions with type **int** and
11279 distinct values. The value 0 is reserved for use as the null signal (see *kill()*). Additional
11280 implementation-defined signals may occur in the system.

11281 The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,
11282 SIGSEGV, and SIGTERM to be defined. An implementation need not generate any of these six
11283 CX signals, except as a result of explicit use of interfaces that generate signals, such as *raise()*, *kill()*,
11284 the General Terminal Interface (see [Section 11.1.9](#), on page 204), and the *kill* utility, unless
11285 otherwise stated (see, for example, XSH [Section 2.8.3.3](#), on page 505).

11286 The following signals shall be supported on all implementations (default actions are explained
11287 below the table):

	Signal	Default Action	Description
11288			
11289	SIGABRT	A	Process abort signal.
11290	SIGALRM	T	Alarm clock.
11291	SIGBUS	A	Access to an undefined portion of a memory object.
11292	SIGCHLD	I	Child process terminated, stopped,
11293	XSI		or continued.
11294	SIGCONT	C	Continue executing, if stopped.
11295	SIGFPE	A	Erroneous arithmetic operation.
11296	SIGHUP	T	Hangup.
11297	SIGILL	A	Illegal instruction.
11298	SIGINT	T	Terminal interrupt signal.
11299	SIGKILL	T	Kill (cannot be caught or ignored).
11300	SIGPIPE	T	Write on a pipe with no one to read it.
11301	SIGQUIT	A	Terminal quit signal.
11302	SIGSEGV	A	Invalid memory reference.
11303	SIGSTOP	S	Stop executing (cannot be caught or ignored).
11304	SIGTERM	T	Termination signal.
11305	SIGTSTP	S	Terminal stop signal.
11306	SIGTTIN	S	Background process attempting read.
11307	SIGTTOU	S	Background process attempting write.
11308	SIGUSR1	T	User-defined signal 1.
11309	SIGUSR2	T	User-defined signal 2.
11310	OB XSR	T	Pollable event.
11311	OB XSI	T	Profiling timer expired.
11312	XSI	A	Bad system call.
11313		A	Trace/breakpoint trap.
11314		I	High bandwidth data is available at a socket.
11315	XSI	T	Virtual timer expired.
11316		A	CPU time limit exceeded.
11317		A	File size limit exceeded.

11318 The default actions are as follows:

- 11319 T Abnormal termination of the process.
- 11320 XSI A Abnormal termination of the process with additional actions.
- 11321 I Ignore the signal.
- 11322 S Stop the process.
- 11323 C Continue the process, if it is stopped; otherwise, ignore the signal.

11324 The effects on the process in each case are described in XSH Section 2.4.3 (on page 490).

11325 CX The <signal.h> header shall declare the **sigaction** structure, which shall include at least the following members:

```

11327 void (*sa_handler)(int) Pointer to a signal-catching function
11328 or one of the SIG_IGN or SIG_DFL.
11329 sigset_t sa_mask Set of signals to be blocked during execution
11330 of the signal handling function.
11331 int sa_flags Special flags.
11332 void (*sa_sigaction)(int, siginfo_t *, void *)
11333 Pointer to a signal-catching function.

```

11334 CX The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
 11335 shall not use both simultaneously.

11336 The <signal.h> header shall define the following macros which shall expand to integer constant
 11337 expressions that need not be usable in #if preprocessing directives:

11338 CX SIG_BLOCK The resulting set is the union of the current set and the signal set pointed
 11339 to by the argument *set*.

11340 CX SIG_UNBLOCK The resulting set is the intersection of the current set and the complement
 11341 of the signal set pointed to by the argument *set*.

11342 CX SIG_SETMASK The resulting set is the signal set pointed to by the argument *set*.

11343 The <signal.h> header shall also define the following symbolic constants:

11344 CX SA_NOCLDSTOP Do not generate SIGCHLD when children stop
 11345 XSI or stopped children continue.

11346 XSI SA_ONSTACK Causes signal delivery to occur on an alternate stack.

11347 CX SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry to signal
 11348 handlers.

11349 CX SA_RESTART Causes certain functions to become restartable.

11350 CX SA_SIGINFO Causes extra information to be passed to signal handlers at the time of
 11351 receipt of a signal.

11352 XSI SA_NOCLDWAIT Causes implementations not to create zombie processes or status
 11353 information on child termination. See *sigaction()*.

11354 CX SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.

11355 XSI SS_ONSTACK Process is executing on an alternate signal stack.

11356 XSI SS_DISABLE Alternate signal stack is disabled.

11357 XSI MINSIGSTKSZ Minimum stack size for a signal handler.

11358 XSI SIGSTKSZ Default size in bytes for the alternate signal stack.

11359 CX The <signal.h> header shall define the **mcontext_t** type through **typedef**.

11360 CX The <signal.h> header shall define the **ucontext_t** type as a structure that shall include at least
 11361 the following members:

11362	<code>ucontext_t *uc_link</code>	Pointer to the context that is resumed when this context returns.
11363		
11364	<code>sigset_t uc_sigmask</code>	The set of signals that are blocked when this context is active.
11365		
11366	<code>stack_t uc_stack</code>	The stack used by this context.
11367	<code>mcontext_t uc_mcontext</code>	A machine-specific representation of the saved context.
11368		

11369 The <signal.h> header shall define the **stack_t** type as a structure, which shall include at least
 11370 the following members:

11371	<code>void *ss_sp</code>	Stack base or pointer.
11372	<code>size_t ss_size</code>	Stack size.
11373	<code>int ss_flags</code>	Flags.

11374 CX The <signal.h> header shall define the **siginfo_t** type as a structure, which shall include at least
11375 the following members:

11376	CX	int	si_signo	Signal number.
11377		int	si_code	Signal code.
11378	XSI	int	si_errno	If non-zero, an <i>errno</i> value associated with 11379 this signal, as described in <errno.h>.
11380	CX	pid_t	si_pid	Sending process ID.
11381		uid_t	si_uid	Real user ID of sending process.
11382		void	*si_addr	Address of faulting instruction.
11383		int	si_status	Exit value or signal.
11384	OB XSR	long	si_band	Band event for SIGPOLL.
11385	CX	union sigval	si_value	Signal value.

11386 CX The <signal.h> header shall define the symbolic constants in the **Code** column of the following
11387 table for use as values of *si_code* that are signal-specific or non-signal-specific reasons why the
11388 signal was generated.

	Signal	Code	Reason
11389	SIGILL CX	ILL_ILLOPC	Illegal opcode.
11390		ILL_ILLOPN	Illegal operand.
11391		ILL_ILLADR	Illegal addressing mode.
11392		ILL_ILLTRP	Illegal trap.
11393		ILL_PRVOPC	Privileged opcode.
11394		ILL_PRVREG	Privileged register.
11395		ILL_COPROC	Coprocessor error.
11396		ILL_BADSTK	Internal stack error.
11397		SIGFPE	FPE_INTDIV
11398	FPE_INTOVF		Integer overflow.
11399	FPE_FLTDIV		Floating-point divide by zero.
11400	FPE_FLTOVF		Floating-point overflow.
11401	FPE_FLTUND		Floating-point underflow.
11402	FPE_FLTRES		Floating-point inexact result.
11403	FPE_FLTINV		Invalid floating-point operation.
11404	FPE_FLTSUB		Subscript out of range.
11405	SIGSEGV	SEGV_MAPERR	Address not mapped to object.
11406		SEGV_ACCERR	Invalid permissions for mapped object.
11407	SIGBUS	BUS_ADRALN	Invalid address alignment.
11408		BUS_ADRERR	Nonexistent physical address.
11409		BUS_OBJERR	Object-specific hardware error.
11410	SIGTRAP	TRAP_BRKPT	Process breakpoint.
11411		TRAP_TRACE	Process trace trap.
11412	SIGCHLD CX	CLD_EXITED	Child has exited.
11413		CLD_KILLED	Child has terminated abnormally and did not create a core file.
11414		CLD_DUMPED	Child has terminated abnormally and created a core file.
11415		CLD_TRAPPED	Traced child has trapped.
11416		CLD_STOPPED	Child has stopped.
11417		CLD_CONTINUED	Stopped child has continued.
11418	SIGPOLL OB XSR	POLL_IN	Data input available.
11419		POLL_OUT	Output buffers available.
11420		POLL_MSG	Input message available.
11421		POLL_ERR	I/O error.
11422		POLL_PRI	High priority input available.
11423		POLL_HUP	Device disconnected.
11424	Any CX	SI_USER	Signal sent by <i>kill()</i> .
11425		SI_QUEUE	Signal sent by <i>sigqueue()</i> .
11426		SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .
11427		SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.
11428		SI_MESGQ	Signal generated by arrival of a message on an empty message queue
11429			
11430			
11431			

11432 CX Implementations may support additional *si_code* values not included in this list, may generate values included in this list under circumstances other than those described in this list, and may contain extensions or limitations that prevent some values from being generated. Implementations do not generate a different value from the ones described in this list for circumstances described in this list.

11437 CX In addition, the following signal-specific information shall be available:

Signal	Member	Value
11438 11439 11440 SIGILL SIGFPE	void * <i>si_addr</i>	Address of faulting instruction.
11441 11442 SIGSEGV SIGBUS	void * <i>si_addr</i>	Address of faulting memory reference.
11443 11444 11445 11446 11447 11448 11449 11450 SIGCHLD	pid_t <i>si_pid</i> int <i>si_status</i> uid_t <i>si_uid</i>	Child process ID. If <i>si_code</i> is equal to CLD_EXITED, then <i>si_status</i> holds the exit value of the process; otherwise, it is equal to the signal that caused the process to change state. The exit value in <i>si_status</i> shall be equal to the full exit value (that is, the value passed to <i>_exit()</i> , <i>_Exit()</i> , or <i>exit()</i> , or returned from <i>main()</i>); it shall not be limited to the least significant eight bits of the value. Real user ID of the process that sent the signal.
11451 OB XSR SIGPOLL	long <i>si_band</i>	Band event for POLL_IN, POLL_OUT, or POLL_MSG

11452 For some implementations, the value of *si_addr* may be inaccurate.

11453 The following shall be declared as functions and may also be defined as macros. Function
11454 prototypes shall be provided.

```

11455 CX int kill(pid_t, int);
11456 XSI int killpg(pid_t, int);
11457 CX void psiginfo(const siginfo_t *, const char *);
11458 void psignal(int, const char *);
11459 int pthread_kill(pthread_t, int);
11460 int pthread_sigmask(int, const sigset_t *restrict,
11461 sigset_t *restrict);
11462 int raise(int);
11463 CX int sigaction(int, const struct sigaction *restrict,
11464 struct sigaction *restrict);
11465 int sigaddset(sigset_t *, int);
11466 XSI int sigaltstack(const stack_t *restrict, stack_t *restrict);
11467 CX int sigdelset(sigset_t *, int);
11468 int sigemptyset(sigset_t *);
11469 int sigfillset(sigset_t *);
11470 OB XSI int sighold(int);
11471 int sigignore(int);
11472 int siginterrupt(int, int);
11473 CX int sigismember(const sigset_t *, int);
11474 void (*signal(int, void (*)(int)))(int);
11475 OB XSI int sigpause(int);
11476 CX int sigpending(sigset_t *);
11477 int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
11478 int sigqueue(pid_t, int, union sigval);
11479 OB XSI int sigrelse(int);
11480 void (*sigset(int, void (*)(int)))(int);
11481 CX int sigsuspend(const sigset_t *);
11482 int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
11483 const struct timespec *restrict);
11484 int sigwait(const sigset_t *restrict, int *restrict);
11485 int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);

```

11486 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

11487 APPLICATION USAGE

11488 On systems not supporting the XSI option, the *si_pid* and *si_uid* members of **siginfo_t** are only
 11489 required to be valid when *si_code* is SI_USER or SI_QUEUE. On XSI-conforming systems, they
 11490 are also valid for all *si_code* values less than or equal to 0; however, it is unspecified whether
 11491 SI_USER and SI_QUEUE have values less than or equal to zero, and therefore XSI applications
 11492 should check whether *si_code* has the value SI_USER or SI_QUEUE or is less than or equal to 0 to
 11493 tell whether *si_pid* and *si_uid* are valid.

11494 RATIONALE

11495 None.

11496 FUTURE DIRECTIONS

11497 The SIGPOLL and SIGPROF signals may be removed in a future version.

11498 SEE ALSO

11499 <errno.h>, <stropts.h>, <sys/types.h>, <time.h>

11500 XSH Section 2.2 (on page 472), *alarm()*, *ioctl()*, *kill()*, *killpg()*, *psiginfo()*, *pthread_kill()*,
 11501 *pthread_sigmask()*, *raise()*, *sigaction()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*,
 11502 *sigfillset()*, *sighold()*, *siginterrupt()*, *sigismember()*, *signal()*, *sigpending()*, *sigqueue()*, *sigsuspend()*,
 11503 *sigtimedwait()*, *sigwait()*, *timer_create()*, *wait()*, *waitid()*

11504 XCU *kill*

11505 CHANGE HISTORY

11506 First released in Issue 1.

11507 Issue 5

11508 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 11509 Threads Extension.

11510 The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is
 11511 removed.

11512 Issue 6

11513 The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for
 11514 abnormal termination is clarified.

11515 The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()*
 11516 function.

11517 The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function*
 11518 function member of the **sigevent** structure.

11519 The following new requirements on POSIX implementations derive from alignment with the
 11520 Single UNIX Specification:

11521 The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now
 11522 mandated. This is also a FIPS requirement.

11523 The **pid_t** definition is mandated.

11524 The RT markings are changed to RTS to denote that the semantics are part of the Realtime
 11525 Signals Extension option.

11526 The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*,
 11527 *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

- 11528 IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from
11529 **<time.h>** may be made visible when **<signal.h>** is included.
- 11530 Extensions beyond the ISO C standard are marked.
- 11531 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive
11532 text for members of the **sigaction** structure.
- 11533 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of
11534 the *sa_sigaction* member of the **sigaction** structure.
- 11535 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of
11536 the **siginfo_t** type structure in the DESCRIPTION. This is an editorial change and no normative
11537 change is intended.
- 11538 **Issue 7**
- 11539 SD5-XBD-ERN-5 is applied.
- 11540 SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed
11541 at the same time as the LEGACY *sigstack()* function.
- 11542 SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.
- 11543 Austin Group Interpretation 1003.1-2001 #034 is applied.
- 11544 The **ucontext_t** and **mcontext_t** structures are added here from the obsolescent **<ucontext.h>**
11545 header.
- 11546 The *psiginfo()* and *psignal()* functions are added from The Open Group Technical Standard, 2006,
11547 Extended API Set Part 1.
- 11548 The SIGPOLL and SIGPROF signals and text relating to the XSI STREAMS option are marked
11549 obsolescent.
- 11550 The SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLDWAIT, and SA_NODEFER
11551 constants are moved from the XSI option to the Base.
- 11552 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 11553 This reference page is clarified with respect to macros and symbolic constants, and declarations
11554 for the **pthread_attr_t**, **pthread_t**, and **uid_t** types and the **timespec** structure are added.
- 11555 SIGRTMIN and SIGRTMAX are required to be positive integer expressions.
- 11556 The APPLICATION USAGE section is updated to describe the *si_pid* and *si_uid* members of
11557 **siginfo_t**.
- 11558 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0062 [208], XBD/TC1-2008/0063 [80],
11559 and XBD/TC1-2008/0064 [157] are applied.
- 11560 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0070 [536], XBD/TC2-2008/0071 [690],
11561 XBD/TC2-2008/0072 [594], XBD/TC2-2008/0073 [844], and XBD/TC2-2008/0074 [536] are
11562 applied.

11563 **NAME**
 11564 spawn.h ‡(spawn,ADVANCED REALTIME)

11565 **SYNOPSIS**

11566 SPN #include <spawn.h>

11567 **DESCRIPTION**

11568 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
 11569 types used in performing spawn operations.

11570 The <spawn.h> header shall define the **mode_t** and **pid_t** types as described in <sys/types.h>.

11571 The <spawn.h> header shall define the **sigset_t** type as described in <signal.h>.

11572 The tag **sched_param** shall be declared as naming an incomplete structure type, the contents of
 11573 which are described in the <sched.h> header.

11574 The <spawn.h> header shall define the following symbolic constants for use as the flags that
 11575 may be set in a **posix_spawnattr_t** object using the *posix_spawnattr_setflags()* function:

11576 POSIX_SPAWN_RESETEIDS
 11577 POSIX_SPAWN_SETPGROUP
 11578 PS POSIX_SPAWN_SETSCHEDPARAM
 11579 POSIX_SPAWN_SETSCHEDULER
 11580 POSIX_SPAWN_SETSIGDEF
 11581 POSIX_SPAWN_SETSIGMASK

11582 The following shall be declared as functions and may also be defined as macros. Function
 11583 prototypes shall be provided.

```

11584 int  posix_spawn(pid_t *restrict, const char *restrict,
11585                const posix_spawn_file_actions_t *,
11586                const posix_spawnattr_t *restrict, char *const [restrict],
11587                char *const [restrict]);
11588 int  posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11589                int);
11590 int  posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11591                int, int);
11592 int  posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11593                int, const char *restrict, int, mode_t);
11594 int  posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11595 int  posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11596 int  posix_spawnattr_destroy(posix_spawnattr_t *);
11597 int  posix_spawnattr_getflags(const posix_spawnattr_t *restrict,
11598                short *restrict);
11599 int  posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11600                pid_t *restrict);
11601 PS  int  posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11602                struct sched_param *restrict);
11603 int  posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11604                int *restrict);
11605 int  posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,
11606                sigset_t *restrict);
11607 int  posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,
11608                sigset_t *restrict);
11609 int  posix_spawnattr_init(posix_spawnattr_t *);
    
```

```

11610     int    posix_spawnattr_setflags(posix_spawnattr_t *, short);
11611     int    posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);
11612 PS    int    posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11613         const struct sched_param *restrict);
11614     int    posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11615     int    posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,
11616         const sigset_t *restrict);
11617     int    posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11618         const sigset_t *restrict);
11619     int    posix_spawn(pid_t *restrict, const char *restrict,
11620         const posix_spawn_file_actions_t *,
11621         const posix_spawnattr_t *restrict,
11622         char *const [restrict], char *const [restrict]);

```

11623 Inclusion of the **<spawn.h>** header may make visible symbols defined in the **<sched.h>** and
 11624 **<signal.h>** headers.

11625 APPLICATION USAGE

11626 None.

11627 RATIONALE

11628 None.

11629 FUTURE DIRECTIONS

11630 None.

11631 SEE ALSO

11632 [<sched.h>](#), [<semaphore.h>](#), [<signal.h>](#), [<sys/types.h>](#)

11633 XSH [posix_spawn\(\)](#), [posix_spawn_file_actions_addclose\(\)](#), [posix_spawn_file_actions_adddup2\(\)](#),
 11634 [posix_spawn_file_actions_destroy\(\)](#), [posix_spawnattr_destroy\(\)](#), [posix_spawnattr_getflags\(\)](#),
 11635 [posix_spawnattr_getpgroup\(\)](#), [posix_spawnattr_getschedparam\(\)](#), [posix_spawnattr_getschedpolicy\(\)](#),
 11636 [posix_spawnattr_getsigdefault\(\)](#), [posix_spawnattr_getsigmask\(\)](#)

11637 CHANGE HISTORY

11638 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11639 The **restrict** keyword is added to the prototypes for [posix_spawn\(\)](#),
 11640 [posix_spawn_file_actions_addopen\(\)](#), [posix_spawnattr_getsigdefault\(\)](#), [posix_spawnattr_getflags\(\)](#),
 11641 [posix_spawnattr_getpgroup\(\)](#), [posix_spawnattr_getschedparam\(\)](#), [posix_spawnattr_getschedpolicy\(\)](#),
 11642 [posix_spawnattr_getsigmask\(\)](#), [posix_spawnattr_setsigdefault\(\)](#), [posix_spawnattr_setschedparam\(\)](#),
 11643 [posix_spawnattr_setsigmask\(\)](#), and [posix_spawnnp\(\)](#).

11644 Issue 7

11645 This reference page is clarified with respect to macros and symbolic constants, and declarations
 11646 for the **mode_t**, **pid_t**, and **sigset_t** types are added.

11647 **NAME**

11648 stdarg.h — handle variable argument list

11649 **SYNOPSIS**

```
11650        #include <stdarg.h>
11651        void va_start(va_list ap, argN);
11652        void va_copy(va_list dest, va_list src);
11653        type va_arg(va_list ap, type);
11654        void va_end(va_list ap);
```

11655 **DESCRIPTION**

11656 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11657 conflict between the requirements described here and the ISO C standard is unintentional. This
11658 volume of POSIX.1-2017 defers to the ISO C standard.

11659 The <stdarg.h> header shall contain a set of macros which allows portable functions that accept
11660 variable argument lists to be written. Functions that have variable argument lists (such as
11661 *printf()*) but do not use these macros are inherently non-portable, as different systems use
11662 different argument-passing conventions.

11663 The <stdarg.h> header shall define the **va_list** type for variables used to traverse the list.

11664 The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to
11665 *va_arg()*.

11666 The *va_copy()* macro initializes *dest* as a copy of *src*, as if the *va_start()* macro had been applied
11667 to *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used
11668 to reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to
11669 reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

11670 The object *ap* may be passed as an argument to another function; if that function invokes the
11671 *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall
11672 be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is the
11673 identifier of the rightmost parameter in the variable parameter list in the function definition (the
11674 one just before the *...*). If the parameter *argN* is declared with the **register** storage class, with a
11675 function type or array type, or with a type that is not compatible with the type that results after
11676 application of the default argument promotions, the behavior is undefined.

11677 The *va_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation
11678 of *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*
11679 parameter shall be a type name specified such that the type of a pointer to an object that has the
11680 specified type can be obtained simply by postfixing a '*' to type. If there is no actual next
11681 argument, or if *type* is not compatible with the type of the actual next argument (as promoted
11682 according to the default argument promotions), the behavior is undefined, except for the
11683 following cases:

11684 One type is a signed integer type, the other type is the corresponding unsigned integer
11685 type, and the value is representable in both types.

11686 One type is a pointer to **void** and the other is a pointer to a character type.

11687 XSI Both types are pointers.

11688 Different types can be mixed, but it is up to the routine to know what type of argument is
11689 expected.

11690 The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is
11691 invoked again).

11692 Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding
11693 invocation of the *va_end()* macro in the same function.

11694 Multiple traversals, each bracketed by *va_start()* ... *va_end()*, are possible.

11695 EXAMPLES

11696 This example is a possible implementation of *execl()*:

```
11697 #include <stdarg.h>
11698 #define MAXARGS 31
11699 /*
11700  * execl is called by
11701  * execl(file, arg1, arg2, ..., (char *) (0));
11702  */
11703 int execl(const char *file, const char *args, ...)
11704 {
11705     va_list ap;
11706     char *array[MAXARGS + 1];
11707     int argno = 0;
11708
11709     va_start(ap, args);
11710     while (args != 0 && argno < MAXARGS)
11711     {
11712         array[argno++] = args;
11713         args = va_arg(ap, const char *);
11714     }
11715     array[argno] = (char *) 0;
11716     va_end(ap);
11717     return execv(file, array);
11718 }
```

11718 APPLICATION USAGE

11719 It is up to the calling routine to communicate to the called routine how many arguments there
11720 are, since it is not always possible for the called routine to determine this in any other way. For
11721 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell
11722 how many arguments are there by the *format* argument.

11723 RATIONALE

11724 None.

11725 FUTURE DIRECTIONS

11726 None.

11727 SEE ALSO

11728 XSH *exec*, *fprintf()*

11729 CHANGE HISTORY

11730 First released in Issue 4. Derived from the ANSI C standard.

11731 Issue 6

11732 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

11733 **NAME**

11734 stdbool.h ‡boolean type and values

11735 **SYNOPSIS**

11736 #include <stdbool.h>

11737 **DESCRIPTION**

11738 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11739 conflict between the requirements described here and the ISO C standard is unintentional. This
11740 volume of POSIX.1-2017 defers to the ISO C standard.

11741 The <stdbool.h> header shall define the following macros:

11742 bool Expands to **_Bool**.

11743 true Expands to the integer constant 1.

11744 false Expands to the integer constant 0.

11745 __bool_true_false_are_defined

11746 Expands to the integer constant 1.

11747 An application may undefine and then possibly redefine the macros bool, true, and false.

11748 **APPLICATION USAGE**

11749 None.

11750 **RATIONALE**

11751 None.

11752 **FUTURE DIRECTIONS**

11753 The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature
11754 and may be removed in a future version.

11755 **SEE ALSO**

11756 None.

11757 **CHANGE HISTORY**

11758 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11759 **NAME**11760 **stddef.h** †'standard type definitions11761 **SYNOPSIS**

11762 #include <stddef.h>

11763 **DESCRIPTION**

11764 CX The functionality described on this reference page is aligned with the ISO C standard. Any
11765 conflict between the requirements described here and the ISO C standard is unintentional. This
11766 volume of POSIX.1-2017 defers to the ISO C standard.

11767 The **<stddef.h>** header shall define the following macros:

11768 CX **NULL** Null pointer constant. The macro shall expand to an integer constant expression
11769 with the value 0 cast to type **void ***.

11770 offsetof(*type*, *member-designator*)

11771 Integer constant expression of type **size_t**, the value of which is the offset in bytes
11772 to the structure member (*member-designator*), from the beginning of its structure
11773 (*type*).

11774 The **<stddef.h>** header shall define the following types:11775 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11776 **wchar_t** Integer type whose range of values can represent distinct codes for all members of
11777 the largest extended character set specified among the supported locales; the null
11778 character shall have the code value zero. Each member of the basic character set
11779 shall have a code value equal to its value when used as the lone character in an
11780 integer character constant if an implementation does not define
11781 `__STDC_MB_MIGHT_NEQ_WC__`.

11782 **size_t** Unsigned integer type of the result of the *sizeof* operator.

11783 The implementation shall support one or more programming environments in which the widths
11784 of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these
11785 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

11786 **APPLICATION USAGE**

11787 None.

11788 **RATIONALE**

11789 The ISO C standard does not require the **NULL** macro to include the cast to type **void *** and
11790 specifies that the **NULL** macro be implementation-defined. POSIX.1-2017 requires the cast and
11791 therefore need not be implementation-defined.

11792 **FUTURE DIRECTIONS**

11793 None.

11794 **SEE ALSO**11795 [<sys/types.h>](#), [<wchar.h>](#)11796 XSH *confstr()*11797 XCU *getconf*11798 **CHANGE HISTORY**

11799 First released in Issue 4. Derived from the ANSI C standard.

11800 **Issue 7**

11801 This reference page is clarified with respect to macros and symbolic constants.

11802 SD5-XBD-ERN-53 is applied, updating the definition of `wchar_t` to align with
11803 ISO/IEC 9899:1999 standard, Technical Corrigendum 3.

11804 **NAME**

11805 stdint.h ‡integer types

11806 **SYNOPSIS**

11807 #include <stdint.h>

11808 **DESCRIPTION**

11809 CX Some of the functionality described on this reference page extends the ISO C standard.
 11810 Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 472) to
 11811 enable the visibility of these symbols in this header.

11812 The **<stdint.h>** header shall declare sets of integer types having specified widths, and shall
 11813 define corresponding sets of macros. It shall also define macros that specify limits of integer
 11814 types corresponding to types defined in other standard headers.

11815 **Note:** The “width” of an integer type is the number of bits used to store its value in a pure binary
 11816 system; the actual type may use more bits than that (for example, a 28-bit type could be stored
 11817 in 32 bits of actual storage). An N -bit signed type has values in the range -2^{N-1} or $1-2^{N-1}$ to
 11818 $2^{N-1}-1$, while an N -bit unsigned type has values in the range 0 to 2^N-1 .

11819 Types are defined in the following categories:

- 11820 Integer types having certain exact widths
- 11821 Integer types having at least certain specified widths
- 11822 Fastest integer types having at least certain specified widths
- 11823 Integer types wide enough to hold pointers to objects
- 11824 Integer types having greatest width

11825 (Some of these types may denote the same type.)

11826 Corresponding macros specify limits of the declared types and construct suitable constants.

11827 For each type described herein that the implementation provides, the **<stdint.h>** header shall
 11828 declare that **typedef** name and define the associated macros. Conversely, for each type described
 11829 herein that the implementation does not provide, the **<stdint.h>** header shall not declare that
 11830 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
 11831 types described as required, but need not provide any of the others (described as optional).

11832 **Integer Types**

11833 When **typedef** names differing only in the absence or presence of the initial u are defined, they
 11834 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
 11835 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also
 11836 provide the other.

11837 In the following descriptions, the symbol N represents an unsigned decimal integer with no
 11838 leading zeros (for example, 8 or 24, but not 04 or 048).

11839 Exact-width integer types

11840 The **typedef** name **int N _t** designates a signed integer type with width N , no padding bits,
 11841 and a two’s-complement representation. Thus, **int8_t** denotes a signed integer type with a
 11842 width of exactly 8 bits.

11843 The **typedef** name **uint N _t** designates an unsigned integer type with width N . Thus,
 11844 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

11845 CX The following types are required:

11846 **int8_t**

11847 **int16_t**

11848 **int32_t**

11849 **uint8_t**

11850 **uint16_t**

11851 **uint32_t**

11852 If an implementation provides integer types with width 64 that meet these requirements,
 11853 then the following types are required:

11854 **int64_t**

11855 **uint64_t**

11856 CX In particular, this will be the case if any of the following are true:

11857 † If implementation supports the `_POSIX_V7_ILP32_OFFBIG` programming
 11858 environment and the application is being built in the `_POSIX_V7_ILP32_OFFBIG`
 11859 programming environment (see the Shell and Utilities volume of POSIX.1-2017, c99,
 11860 Programming Environments).

11861 † If implementation supports the `_POSIX_V7_LP64_OFF64` programming
 11862 environment and the application is being built in the `_POSIX_V7_LP64_OFF64`
 11863 programming environment.

11864 † If implementation supports the `_POSIX_V7_LPBIG_OFFBIG` programming
 11865 environment and the application is being built in the `_POSIX_V7_LPBIG_OFFBIG`
 11866 programming environment.

11867 All other types of this form are optional.

11868 Minimum-width integer types

11869 The **typedef** name **int_leastN_t** designates a signed integer type with a width of at least *N*,
 11870 such that no signed integer type with lesser size has at least the specified width. Thus,
 11871 **int_least32_t** denotes a signed integer type with a width of at least 32 bits.

11872 The **typedef** name **uint_leastN_t** designates an unsigned integer type with a width of at
 11873 least *N*, such that no unsigned integer type with lesser size has at least the specified width.
 11874 Thus, **uint_least16_t** denotes an unsigned integer type with a width of at least 16 bits.

11875 The following types are required:

11876 **int_least8_t**

11877 **int_least16_t**

11878 **int_least32_t**

11879 **int_least64_t**

11880 **uint_least8_t**

11881 **uint_least16_t**

11882 **uint_least32_t**

11883 **uint_least64_t**

11884 All other types of this form are optional.

11885 Fastest minimum-width integer types

11886 Each of the following types designates an integer type that is usually fastest to operate
11887 with among all integer types that have at least the specified width.

11888 The designated type is not guaranteed to be fastest for all purposes; if the implementation
11889 has no clear grounds for choosing one type over another, it will simply pick some integer
11890 type satisfying the signedness and width requirements.

11891 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
11892 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with
11893 a width of at least *N*.

11894 The following types are required:

11895 **int_fast8_t**
11896 **int_fast16_t**
11897 **int_fast32_t**
11898 **int_fast64_t**
11899 **uint_fast8_t**
11900 **uint_fast16_t**
11901 **uint_fast32_t**
11902 **uint_fast64_t**

11903 All other types of this form are optional.

11904 Integer types capable of holding object pointers

11905 The following type designates a signed integer type with the property that any valid
11906 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
11907 the result will compare equal to the original pointer:

11908 **intptr_t**

11909 The following type designates an unsigned integer type with the property that any valid
11910 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
11911 the result will compare equal to the original pointer:

11912 **uintptr_t**

11913 XSI On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required; otherwise, they
11914 are optional.

11915 Greatest-width integer types

11916 The following type designates a signed integer type capable of representing any value of
11917 any signed integer type:

11918 **intmax_t**

11919 The following type designates an unsigned integer type capable of representing any value
11920 of any unsigned integer type:

11921 **uintmax_t**

11922 These types are required.

11923 **Note:** Applications can test for optional types by using the corresponding limit macro from [Limits of](#)
 11924 [Specified-Width Integer Types](#).

11925 **Limits of Specified-Width Integer Types**

11926 The following macros specify the minimum and maximum limits of the types declared in the
 11927 <stdint.h> header. Each macro name corresponds to a similar type name in [Integer Types](#) (on
 11928 page 348).

11929 Each instance of any defined macro shall be replaced by a constant expression suitable for use in
 11930 #if preprocessing directives, and this expression shall have the same type as would an
 11931 expression that is an object of the corresponding type converted according to the integer
 11932 promotions. Its implementation-defined value shall be equal to or greater in magnitude
 11933 (absolute value) than the corresponding value given below, with the same sign, except where
 11934 stated to be exactly the given value.

11935 Limits of exact-width integer types

11936 ‡ ~~Minimum~~ Minimum values of exact-width signed integer types:

11937 {INTN_MIN} Exactly $-(2^{N-1})$

11938 ‡ ~~Maximum~~ Maximum values of exact-width signed integer types:

11939 {INTN_MAX} Exactly $2^{N-1} - 1$

11940 ‡ ~~Maximum~~ Maximum values of exact-width unsigned integer types:

11941 {UINTN_MAX} Exactly $2^N - 1$

11942 Limits of minimum-width integer types

11943 ‡ ~~Minimum~~ Minimum values of minimum-width signed integer types:

11944 {INT_LEASTN_MIN} $-(2^{N-1} - 1)$

11945 ‡ ~~Maximum~~ Maximum values of minimum-width signed integer types:

11946 {INT_LEASTN_MAX} $2^{N-1} - 1$

11947 ‡ ~~Maximum~~ Maximum values of minimum-width unsigned integer types:

11948 {UINT_LEASTN_MAX} $2^N - 1$

11949 Limits of fastest minimum-width integer types

11950 ‡ ~~Minimum~~ Minimum values of fastest minimum-width signed integer types:

11951 {INT_FASTN_MIN} $-(2^{N-1} - 1)$

11952 ‡ ~~Maximum~~ Maximum values of fastest minimum-width signed integer types:

11953 {INT_FASTN_MAX} $2^{N-1} - 1$

11954 ‡ ~~Maximum~~ Maximum values of fastest minimum-width unsigned integer types:

11955 {UINT_FASTN_MAX} $2^N - 1$

11956 Limits of integer types capable of holding object pointers

11957 ‡ ~~Minimum~~ Minimum value of pointer-holding signed integer type:

11958 {INTPTR_MIN} $-(2^{15} - 1)$

11959 ‡ ~~Minimum~~ value of pointer-holding signed integer type:

11960 {INTPTR_MAX} $2^{15} - 1$

11961 ‡ ~~Minimum~~ value of pointer-holding unsigned integer type:

11962 {UINTPTR_MAX} $2^{16} - 1$

11963 Limits of greatest-width integer types

11964 ‡ ~~Minimum~~ value of greatest-width signed integer type:

11965 {INTMAX_MIN} $-(2^{63} - 1)$

11966 ‡ ~~Minimum~~ value of greatest-width signed integer type:

11967 {INTMAX_MAX} $2^{63} - 1$

11968 ‡ ~~Minimum~~ value of greatest-width unsigned integer type:

11969 {UINTMAX_MAX} $2^{64} - 1$

11970 **Limits of Other Integer Types**

11971 The following macros specify the minimum and maximum limits of integer types corresponding
11972 to types defined in other standard headers.

11973 Each instance of these macros shall be replaced by a constant expression suitable for use in #if
11974 preprocessing directives, and this expression shall have the same type as would an expression
11975 that is an object of the corresponding type converted according to the integer promotions. Its
11976 implementation-defined value shall be equal to or greater in magnitude (absolute value) than
11977 the corresponding value given below, with the same sign.

11978 Limits of **ptrdiff_t**:

11979 {PTRDIFF_MIN} -65 535

11980 {PTRDIFF_MAX} +65 535

11981 Limits of **sig_atomic_t**:

11982 {SIG_ATOMIC_MIN} See below.

11983 {SIG_ATOMIC_MAX} See below.

11984 Limit of **size_t**:

11985 {SIZE_MAX} 65 535

11986 Limits of **wchar_t**:

11987 {WCHAR_MIN} See below.

11988 {WCHAR_MAX} See below.

11989 Limits of **wint_t**:

11990 {WINT_MIN} See below.

11991 {WINT_MAX} See below.

11992 If **sig_atomic_t** (see the <signal.h> header) is defined as a signed integer type, the value of
11993 {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall
11994 be no less than 127; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and
11995 the value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no

11996 less than 255.

11997 If **wchar_t** (see the <stddef.h> header) is defined as a signed integer type, the value of
 11998 {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less
 11999 than 127; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of
 12000 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

12001 If **wint_t** (see the <wchar.h> header) is defined as a signed integer type, the value of
 12002 {WINT_MIN} shall be no greater than -32767 and the value of {WINT_MAX} shall be no less
 12003 than 32767; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of
 12004 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65535.

12005 **Macros for Integer Constant Expressions**

12006 The following macros expand to integer constant expressions suitable for initializing objects that
 12007 have integer types corresponding to types defined in the <stdint.h> header. Each macro name
 12008 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*
 12009 *integer types*.

12010 Each invocation of one of these macros shall expand to an integer constant expression suitable
 12011 for use in #if preprocessing directives. The type of the expression shall have the same type as
 12012 would an expression that is an object of the corresponding type converted according to the
 12013 integer promotions. The value of the expression shall be that of the argument.

12014 The argument in any instance of these macros shall be an unsuffixed integer constant with a
 12015 value that does not exceed the limits for the corresponding type.

12016 **Macros for minimum-width integer constant expressions**

12017 The macro *INTN_C(value)* shall expand to an integer constant expression corresponding to
 12018 the type **int_leastN_t**. The macro *UINTN_C(value)* shall expand to an integer constant
 12019 expression corresponding to the type **uint_leastN_t**. For example, if **uint_least64_t** is a
 12020 name for the type **unsigned long long**, then *UINT64_C(0x123)* might expand to the integer
 12021 constant 0x123ULL.

12022 **Macros for greatest-width integer constant expressions**

12023 The following macro expands to an integer constant expression having the value specified
 12024 by its argument and the type **intmax_t**:

12025 *INTMAX_C(value)*

12026 The following macro expands to an integer constant expression having the value specified
 12027 by its argument and the type **uintmax_t**:

12028 *UINTMAX_C(value)*

12029 **APPLICATION USAGE**

12030 None.

12031 **RATIONALE**

12032 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in
 12033 freestanding environments, which might not support the formatted I/O functions. In some
 12034 environments, if the formatted conversion support is not wanted, using this header instead of
 12035 the <inttypes.h> header avoids defining such a large number of macros.

- 12036 As a consequence of adding **int8_t**, the following are true:
- 12037 A byte is exactly 8 bits.
- 12038 {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the
12039 value -128, and {UCHAR_MAX} has the value 255.
- 12040 (The POSIX standard explicitly requires 8-bit char and two's-complement arithmetic.)
- 12041 **FUTURE DIRECTIONS**
- 12042 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
12043 in the **<stdint.h>** header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
12044 **_MIN**, or **_C** may be added to the macros defined in the **<stdint.h>** header.
- 12045 **SEE ALSO**
- 12046 [<inttypes.h>](#), [<signal.h>](#), [<stddef.h>](#), [<wchar.h>](#)
- 12047 XSH [Section 2.2](#) (on page 472)
- 12048 **CHANGE HISTORY**
- 12049 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.
- 12050 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is applied.
- 12051 **Issue 7**
- 12052 ISO/IEC 9899:1999 standard, Technical Corrigendum 3 #40 is applied.
- 12053 SD5-XBD-ERN-67 is applied.

12054 **NAME**
 12055 stdio.h ‡standard buffered input/output

12056 **SYNOPSIS**
 12057 #include <stdio.h>

12058 **DESCRIPTION**
 12059 CX Some of the functionality described on this reference page extends the ISO C standard.
 12060 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 12061 enable the visibility of these symbols in this header.

12062 The <stdio.h> header shall define the following data types through **typedef**:

12063 **FILE** A structure containing information about a file.

12064 **fpos_t** A non-array type containing all information needed to specify uniquely
 12065 every position within a file.

12066 **off_t** As described in <sys/types.h>.

12067 **size_t** As described in <stddef.h>.

12068 CX **ssize_t** As described in <sys/types.h>.

12069 CX **va_list** As described in <stdarg.h>.

12070 The <stdio.h> header shall define the following macros which shall expand to integer constant
 12071 expressions:

12072 CX **BUFSIZ** Size of <stdio.h> buffers. This shall expand to a positive value.

12073 CX **L_ctermid** Maximum size of character array to hold *ctermid()* output.

12074 OB **L_tmpnam** Maximum size of character array to hold *tmpnam()* output.

12075 The <stdio.h> header shall define the following macros which shall expand to integer constant
 12076 expressions with distinct values:

12077 **_IOFBF** Input/output fully buffered.

12078 **_IOLBF** Input/output line buffered.

12079 **_IONBF** Input/output unbuffered.

12080 The <stdio.h> header shall define the following macros which shall expand to integer constant
 12081 expressions with distinct values:

12082 **SEEK_CUR** Seek relative to current position.

12083 **SEEK_END** Seek relative to end-of-file.

12084 **SEEK_SET** Seek relative to start-of-file.

12085 The <stdio.h> header shall define the following macros which shall expand to integer constant
 12086 expressions denoting implementation limits:

12087 **{FILENAME_MAX}** Maximum size in bytes of the longest pathname that the implementation
 12088 guarantees can be opened.

12089 **{FOPEN_MAX}** Number of streams which the implementation guarantees can be open
 12090 simultaneously. The value is at least eight.

12091 OB **{TMP_MAX}** Minimum number of unique filenames generated by *tmpnam()*.
 12092 Maximum number of times an application can call *tmpnam()* reliably. The
 12093 value of {TMP_MAX} is at least 25.

12094	OB XSI	On XSI-conformant systems, the value of {TMP_MAX} is at least 10 000.
12095		The <stdio.h> header shall define the following macro which shall expand to an integer constant
12096		expression with type int and a negative value:
12097		EOF End-of-file return value.
12098		The <stdio.h> header shall define NULL as described in <stddef.h> .
12099		The <stdio.h> header shall define the following macro which shall expand to a string constant:
12100	OB XSI	P_tmpdir Default directory prefix for <i>tempnam()</i> .
12101		The <stdio.h> header shall define the following macros which shall expand to expressions of
12102		type “pointer to FILE ” that point to the FILE objects associated, respectively, with the standard
12103		error, input, and output streams:
12104		<i>stderr</i> Standard error output stream.
12105		<i>stdin</i> Standard input stream.
12106		<i>stdout</i> Standard output stream.
12107		The following shall be declared as functions and may also be defined as macros. Function
12108		prototypes shall be provided.
12109		<code>void clearerr(FILE *);</code>
12110	CX	<code>char *ctermid(char *);</code>
12111		<code>int dprintf(int, const char *restrict, ...)</code>
12112		<code>int fclose(FILE *);</code>
12113	CX	<code>FILE *fdopen(int, const char *);</code>
12114		<code>int feof(FILE *);</code>
12115		<code>int ferror(FILE *);</code>
12116		<code>int fflush(FILE *);</code>
12117		<code>int fgetc(FILE *);</code>
12118		<code>int fgetpos(FILE *restrict, fpos_t *restrict);</code>
12119		<code>char *fgets(char *restrict, int, FILE *restrict);</code>
12120	CX	<code>int fileno(FILE *);</code>
12121		<code>void flockfile(FILE *);</code>
12122		<code>FILE *fmemopen(void *restrict, size_t, const char *restrict);</code>
12123		<code>FILE *fopen(const char *restrict, const char *restrict);</code>
12124		<code>int fprintf(FILE *restrict, const char *restrict, ...);</code>
12125		<code>int fputc(int, FILE *);</code>
12126		<code>int fputs(const char *restrict, FILE *restrict);</code>
12127		<code>size_t fread(void *restrict, size_t, size_t, FILE *restrict);</code>
12128		<code>FILE *freopen(const char *restrict, const char *restrict,</code>
12129		<code>FILE *restrict);</code>
12130		<code>int fscanf(FILE *restrict, const char *restrict, ...);</code>
12131		<code>int fseek(FILE *, long, int);</code>
12132	CX	<code>int fseeko(FILE *, off_t, int);</code>
12133		<code>int fsetpos(FILE *, const fpos_t *);</code>
12134		<code>long ftell(FILE *);</code>
12135	CX	<code>off_t ftello(FILE *);</code>
12136		<code>int ftrylockfile(FILE *);</code>
12137		<code>void funlockfile(FILE *);</code>
12138		<code>size_t fwrite(const void *restrict, size_t, size_t, FILE *restrict);</code>
12139		<code>int getc(FILE *);</code>

```

12140     int    getchar(void);
12141 CX   int    getc_unlocked(FILE *);
12142     int    getchar_unlocked(void);
12143     ssize_t getdelim(char **restrict, size_t *restrict, int,
12144                   FILE *restrict);
12145     ssize_t getline(char **restrict, size_t *restrict, FILE *restrict);
12146 OB   char   *gets(char *);
12147 CX   FILE   *open_memstream(char **, size_t *);
12148     int    pclose(FILE *);
12149     void   perror(const char *);
12150 CX   FILE   *popen(const char *, const char *);
12151     int    printf(const char *restrict, ...);
12152     int    putc(int, FILE *);
12153     int    putchar(int);
12154 CX   int    putc_unlocked(int, FILE *);
12155     int    putchar_unlocked(int);
12156     int    puts(const char *);
12157     int    remove(const char *);
12158     int    rename(const char *, const char *);
12159 CX   int    renameat(int, const char *, int, const char *);
12160     void   rewind(FILE *);
12161     int    scanf(const char *restrict, ...);
12162     void   setbuf(FILE *restrict, char *restrict);
12163     int    setvbuf(FILE *restrict, char *restrict, int, size_t);
12164     int    snprintf(char *restrict, size_t, const char *restrict, ...);
12165     int    sprintf(char *restrict, const char *restrict, ...);
12166     int    sscanf(const char *restrict, const char *restrict, ...);
12167 OB XSI char   *tempnam(const char *, const char *);
12168     FILE   *tmpfile(void);
12169 OB   char   *tmpnam(char *);
12170     int    ungetc(int, FILE *);
12171 CX   int    vdprintf(int, const char *restrict, va_list);
12172     int    vfprintf(FILE *restrict, const char *restrict, va_list);
12173     int    vfscanf(FILE *restrict, const char *restrict, va_list);
12174     int    vprintf(const char *restrict, va_list);
12175     int    vscanf(const char *restrict, va_list);
12176     int    vsnprintf(char *restrict, size_t, const char *restrict,
12177                   va_list);
12178     int    vsprintf(char *restrict, const char *restrict, va_list);
12179     int    vsscanf(const char *restrict, const char *restrict, va_list);

```

12180 CX **Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.**

12181 **APPLICATION USAGE**

12182 Since standard I/O streams may use an underlying file descriptor to access the file associated
12183 with a stream, application developers need to be aware that {FOPEN_MAX} streams may not be
12184 available if file descriptors are being used to access files that are not associated with streams.

12185 **RATIONALE**

12186 There is a conflict between the ISO C standard and the POSIX definition of the {TMP_MAX}
12187 macro that is addressed by ISO/IEC 9899:1999 standard, Defect Report 336. The POSIX standard
12188 is in alignment with the public record of the response to the Defect Report. This change has not
12189 yet been published as part of the ISO C standard.

12190 **FUTURE DIRECTIONS**

12191 None.

12192 **SEE ALSO**12193 [<stdarg.h>](#), [<stddef.h>](#), [<sys/types.h>](#)

12194 XSH Section 2.2 (on page 472), [clearerr\(\)](#), [ctermid\(\)](#), [fclose\(\)](#), [fdopen\(\)](#), [feof\(\)](#), [ferror\(\)](#), [fflush\(\)](#),
 12195 [fgetc\(\)](#), [fgetpos\(\)](#), [fgets\(\)](#), [fileno\(\)](#), [flockfile\(\)](#), [fmemopen\(\)](#), [fopen\(\)](#), [fprintf\(\)](#), [fputc\(\)](#), [fputs\(\)](#), [fread\(\)](#),
 12196 [freopen\(\)](#), [fscanf\(\)](#), [fseek\(\)](#), [fsetpos\(\)](#), [ftell\(\)](#), [fwrite\(\)](#), [getc\(\)](#), [getchar\(\)](#), [getc_unlocked\(\)](#), [getdelim\(\)](#),
 12197 [getopt\(\)](#), [gets\(\)](#), [open_memstream\(\)](#), [pclose\(\)](#), [perror\(\)](#), [popen\(\)](#), [putc\(\)](#), [putchar\(\)](#), [puts\(\)](#), [remove\(\)](#),
 12198 [rename\(\)](#), [rewind\(\)](#), [setbuf\(\)](#), [setvbuf\(\)](#), [stdin](#), [system\(\)](#), [tempnam\(\)](#), [tmpfile\(\)](#), [tmpnam\(\)](#), [ungetc\(\)](#),
 12199 [vfprintf\(\)](#), [vfscanf\(\)](#)

12200 **CHANGE HISTORY**

12201 First released in Issue 1. Derived from Issue 1 of the SVID.

12202 **Issue 5**

12203 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12204 Large File System extensions are added.

12205 The constant `L_cuserid` and the external variables `optarg`, `opterr`, `optind`, and `optopt` are marked as
 12206 extensions and LEGACY.12207 The `cuserid()` and `getopt()` functions are marked LEGACY.12208 **Issue 6**12209 The constant `L_cuserid` and the external variables `optarg`, `opterr`, `optind`, and `optopt` are removed
 12210 as they were previously marked LEGACY.12211 The `cuserid()`, `getopt()`, and `getw()` functions are removed as they were previously marked
 12212 LEGACY.

12213 Several functions are marked as part of the Thread-Safe Functions option.

12214 This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the
 12215 description of the `fpos_t` type is now explicitly updated to exclude array types.

12216 Extensions beyond the ISO C standard are marked.

12217 **Issue 7**12218 Austin Group Interpretation 1003.1-2001 #172 is applied, adding rationale about a conflict for the
 12219 definition of `{TMP_MAX}` with the ISO C standard.

12220 SD5-XBD-ERN-99 is applied, adding APPLICATION USAGE.

12221 The `dprintf()`, `fmemopen()`, `getdelim()`, `getline()`, `open_memstream()`, and `vdprintf()` functions are
 12222 added from The Open Group Technical Standard, 2006, Extended API Set Part 1.12223 The `renameat()` function is added from The Open Group Technical Standard, 2006, Extended API
 12224 Set Part 2.12225 The `gets()`, `tmpnam()`, and `tempnam()` functions and the `L_tmpnam` macro are marked
 12226 obsolescent.12227 This reference page is clarified with respect to macros and symbolic constants, and a declaration
 12228 for the `off_t` type is added.

12229 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0065 [291,427] is applied.

12230 **NAME**

12231 stdlib.h †'standard library definitions

12232 **SYNOPSIS**

12233 #include <stdlib.h>

12234 **DESCRIPTION**

12235 CX Some of the functionality described on this reference page extends the ISO C standard.
12236 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
12237 enable the visibility of these symbols in this header.

12238 The <stdlib.h> header shall define the following macros which shall expand to integer constant
12239 expressions:

12240 EXIT_FAILURE Unsuccessful termination for *exit()*; evaluates to a non-zero value.

12241 EXIT_SUCCESS Successful termination for *exit()*; evaluates to 0.

12242 {RAND_MAX} Maximum value returned by *rand()*; at least 32 767.

12243 The <stdlib.h> header shall define the following macro which shall expand to a positive integer
12244 expression with type **size_t**:

12245 {MB_CUR_MAX} Maximum number of bytes in a character specified by the current locale
12246 (category *LC_CTYPE*).

12247 CX In the POSIX locale the value of {MB_CUR_MAX} shall be 1.

12248 The <stdlib.h> header shall define NULL as described in <stddef.h>.

12249 The <stdlib.h> header shall define the following data types through **typedef**:

12250 **div_t** Structure type returned by the *div()* function.

12251 **ldiv_t** Structure type returned by the *ldiv()* function.

12252 **lldiv_t** Structure type returned by the *lldiv()* function.

12253 **size_t** As described in <stddef.h>.

12254 **wchar_t** As described in <stddef.h>.

12255 CX In addition, the <stdlib.h> header shall define the following symbolic constants and macros as
12256 described in <sys/wait.h>:

12257 WEXITSTATUS

12258 WIFEXITED

12259 WIFSIGNALED

12260 WIFSTOPPED

12261 WNOHANG

12262 WSTOPSIG

12263 WTERMSIG

12264 WUNTRACED

12265 The following shall be declared as functions and may also be defined as macros. Function
12266 prototypes shall be provided.

12267 void _Exit(int);

12268 XSI long a64l(const char *);

12269 void abort(void);

12270 int abs(int);

```

12271     int          atexit(void (*)(void));
12272     double       atof(const char *);
12273     int          atoi(const char *);
12274     long         atol(const char *);
12275     long long    atoll(const char *);
12276     void         *bsearch(const void *, const void *, size_t, size_t,
12277                          int (*)(const void *, const void *));
12278     void         *calloc(size_t, size_t);
12279     div_t        div(int, int);
12280 XSI     double       drand48(void);
12281     double       erand48(unsigned short [3]);
12282     void         exit(int);
12283     void         free(void *);
12284     char         *getenv(const char *);
12285 CX     int          getsubopt(char **, char *const *, char **);
12286 XSI     int          grantpt(int);
12287     char         *initstate(unsigned, char *, size_t);
12288     long         jrand48(unsigned short [3]);
12289     char         *l64a(long);
12290     long         labs(long);
12291 XSI     void         lcong48(unsigned short [7]);
12292     ldiv_t       ldiv(long, long);
12293     long long    llabs(long long);
12294     lldiv_t      lldiv(long long, long long);
12295 XSI     long         lrand48(void);
12296     void         *malloc(size_t);
12297     int          mblen(const char *, size_t);
12298     size_t       mbstowcs(wchar_t *restrict, const char *restrict, size_t);
12299     int          mbtowc(wchar_t *restrict, const char *restrict, size_t);
12300 CX     char         *mkdtemp(char *);
12301     int          mkstemp(char *);
12302 XSI     long         mrand48(void);
12303     long         nrand48(unsigned short [3]);
12304 ADV     int          posix_memalign(void **, size_t, size_t);
12305 XSI     int          posix_openpt(int);
12306     char         *ptsname(int);
12307     int          putenv(char *);
12308     void         qsort(void *, size_t, size_t, int (*)(const void *,
12309               const void *));
12310     int          rand(void);
12311 OB CX  int          rand_r(unsigned *);
12312 XSI     long         random(void);
12313     void         *realloc(void *, size_t);
12314 XSI     char         *realpath(const char *restrict, char *restrict);
12315     unsigned short *seed48(unsigned short [3]);
12316 CX     int          setenv(const char *, const char *, int);
12317 XSI     void         setkey(const char *);
12318     char         *setstate(char *);
12319     void         srand(unsigned);
12320 XSI     void         srand48(long);
12321     void         srandom(unsigned);
12322     double       strtod(const char *restrict, char **restrict);

```

```

12323 float      strtod(const char *restrict, char **restrict);
12324 long       strtol(const char *restrict, char **restrict, int);
12325 long double strtold(const char *restrict, char **restrict);
12326 long long  strtoll(const char *restrict, char **restrict, int);
12327 unsigned long strtoul(const char *restrict, char **restrict, int);
12328 unsigned long long
12329 strtoull(const char *restrict, char **restrict, int);
12330 int        system(const char *);
12331 XSI int     unlockpt(int);
12332 CX int     unsetenv(const char *);
12333 size_t     wcstombs(char *restrict, const wchar_t *restrict, size_t);
12334 int        wctomb(char *, wchar_t);

```

12335 CX Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>, <limits.h>, <math.h>, and <sys/wait.h>.

12337 **APPLICATION USAGE**

12338 None.

12339 **RATIONALE**

12340 None.

12341 **FUTURE DIRECTIONS**

12342 None.

12343 **SEE ALSO**

12344 [<limits.h>](#), [<math.h>](#), [<stddef.h>](#), [<sys/types.h>](#), [<sys/wait.h>](#)

12345 XSH Section 2.2 (on page 472), [_Exit\(\)](#), [a64l\(\)](#), [abort\(\)](#), [abs\(\)](#), [atexit\(\)](#), [atof\(\)](#), [atoi\(\)](#), [atol\(\)](#),
12346 [bsearch\(\)](#), [calloc\(\)](#), [div\(\)](#), [drand48\(\)](#), [exit\(\)](#), [free\(\)](#), [getenv\(\)](#), [getsubopt\(\)](#), [grantpt\(\)](#), [initstate\(\)](#), [labs\(\)](#),
12347 [ldiv\(\)](#), [malloc\(\)](#), [mblen\(\)](#), [mbstowcs\(\)](#), [mbtowc\(\)](#), [mkdtemp\(\)](#), [posix_memalign\(\)](#), [posix_openpt\(\)](#),
12348 [ptsname\(\)](#), [putenv\(\)](#), [qsort\(\)](#), [rand\(\)](#), [realloc\(\)](#), [realpath\(\)](#), [setenv\(\)](#), [setkey\(\)](#), [strtod\(\)](#), [strtol\(\)](#),
12349 [strtoul\(\)](#), [system\(\)](#), [unlockpt\(\)](#), [unsetenv\(\)](#), [wcstombs\(\)](#), [wctomb\(\)](#)

12350 **CHANGE HISTORY**

12351 First released in Issue 3.

12352 **Issue 5**

12353 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12354 The [ttyslot\(\)](#) and [valloc\(\)](#) functions are marked LEGACY.

12355 The type of the third argument to [initstate\(\)](#) is changed from **int** to **size_t**. The type of the return value from [setstate\(\)](#) is changed from **char** to **char ***, and the type of the first argument is changed from **char *** to **const char ***.

12358 **Issue 6**

12359 The Open Group Corrigendum U021/1 is applied, correcting the prototype for [realpath\(\)](#) to be consistent with the reference page.

12361 The Open Group Corrigendum U028/13 is applied, correcting the prototype for [putenv\(\)](#) to be consistent with the reference page.

12363 The [rand_r\(\)](#) function is marked as part of the Thread-Safe Functions option.

12364 Function prototypes for [setenv\(\)](#) and [unsetenv\(\)](#) are added.

12365 The [posix_memalign\(\)](#) function is added for alignment with IEEE Std 1003.1d-1999.

12366 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

- 12367 The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.
- 12368 The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.
- 12369 Extensions beyond the ISO C standard are marked.
- 12370 **Issue 7**
- 12371 SD5-XBD-ERN-79 and SD5-XBD-ERN-105 are applied.
- 12372 The LEGACY functions are removed.
- 12373 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.
- 12374
- 12375 The *rand_r()* function is marked obsolescent.
- 12376 This reference page is clarified with respect to macros and symbolic constants.
- 12377 The type of the first argument to *setstate()* is changed from **const char *** to **char ***.
- 12378 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0066 [197] is applied.
- 12379 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0075 [663] is applied.

12380 **NAME**

12381 string.h ‡string operations

12382 **SYNOPSIS**

12383 #include <string.h>

12384 **DESCRIPTION**

12385 CX Some of the functionality described on this reference page extends the ISO C standard.
 12386 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 12387 enable the visibility of these symbols in this header.

12388 The <string.h> header shall define NULL and size_t as described in <stddef.h>.

12389 CX The <string.h> header shall define the locale_t type as described in <locale.h>.

12390 The following shall be declared as functions and may also be defined as macros. Function
 12391 prototypes shall be provided for use with ISO C standard compilers.

12392 XSI void *memcpy(void *restrict, const void *restrict, int, size_t);
 12393 void *memchr(const void *, int, size_t);
 12394 int memcmp(const void *, const void *, size_t);
 12395 void *memcpy(void *restrict, const void *restrict, size_t);
 12396 void *memmove(void *, const void *, size_t);
 12397 void *memset(void *, int, size_t);
 12398 CX char *strcpy(char *restrict, const char *restrict);
 12399 char *strncpy(char *restrict, const char *restrict, size_t);
 12400 char *strcat(char *restrict, const char *restrict);
 12401 char *strchr(const char *, int);
 12402 int strcmp(const char *, const char *);
 12403 int strcoll(const char *, const char *);
 12404 CX int strcoll_l(const char *, const char *, locale_t);
 12405 char *strcpy(char *restrict, const char *restrict);
 12406 size_t strcspn(const char *, const char *);
 12407 CX char *strdup(const char *);
 12408 char *strerror(int);
 12409 CX char *strerror_l(int, locale_t);
 12410 int strerror_r(int, char *, size_t);
 12411 size_t strlen(const char *);
 12412 char *strncat(char *restrict, const char *restrict, size_t);
 12413 int strncmp(const char *, const char *, size_t);
 12414 char *strncpy(char *restrict, const char *restrict, size_t);
 12415 CX char *strndup(const char *, size_t);
 12416 size_t strnlen(const char *, size_t);
 12417 char *strpbrk(const char *, const char *);
 12418 char *strrchr(const char *, int);
 12419 CX char *strsignal(int);
 12420 size_t strspn(const char *, const char *);
 12421 char *strstr(const char *, const char *);
 12422 char *strtok(char *restrict, const char *restrict);
 12423 CX char *strtok_r(char *restrict, const char *restrict, char **restrict);
 12424 size_t strxfrm(char *restrict, const char *restrict, size_t);
 12425 CX size_t strxfrm_l(char *restrict, const char *restrict,
 12426 size_t, locale_t);

12427 CX Inclusion of the **<string.h>** header may also make visible all symbols from **<stddef.h>**.

12428 **APPLICATION USAGE**

12429 None.

12430 **RATIONALE**

12431 None.

12432 **FUTURE DIRECTIONS**

12433 None.

12434 **SEE ALSO**

12435 [<locale.h>](#), [<stddef.h>](#), [<sys/types.h>](#)

12436 XSH Section 2.2 (on page 472), [memccpy\(\)](#), [memchr\(\)](#), [memcmp\(\)](#), [memcpy\(\)](#), [memmove\(\)](#),
12437 [memset\(\)](#), [strcat\(\)](#), [strchr\(\)](#), [strcmp\(\)](#), [strcoll\(\)](#), [strcpy\(\)](#), [strcspn\(\)](#), [strdup\(\)](#), [strerror\(\)](#), [strlen\(\)](#),
12438 [strncat\(\)](#), [strncmp\(\)](#), [strncpy\(\)](#), [strpbrk\(\)](#), [strrchr\(\)](#), [strsignal\(\)](#), [strspn\(\)](#), [strstr\(\)](#), [strtok\(\)](#), [strxfrm\(\)](#)

12439 **CHANGE HISTORY**

12440 First released in Issue 1. Derived from Issue 1 of the SVID.

12441 **Issue 5**

12442 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

12443 **Issue 6**

12444 The [strtok_r\(\)](#) function is marked as part of the Thread-Safe Functions option.

12445 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

12446 The [strerror_r\(\)](#) function is added in response to IEEE PASC Interpretation 1003.1c #39.

12447 **Issue 7**

12448 SD5-XBD-ERN-15 is applied, correcting the prototype for the [strerror_r\(\)](#) function.

12449 The [stpncpy\(\)](#), [stpncpy\(\)](#), [strndup\(\)](#), [strnlen\(\)](#), and [strsignal\(\)](#) functions are added from The Open
12450 Group Technical Standard, 2006, Extended API Set Part 1.

12451 The [strcoll_l\(\)](#), [strerror_l\(\)](#), and [strxfrm_l\(\)](#) functions are added from The Open Group Technical
12452 Standard, 2006, Extended API Set Part 4.

12453 This reference page is clarified with respect to macros and symbolic constants, and a declaration
12454 for the **locale_t** type is added.

12455 **NAME**

12456 strings.h ‡string operations

12457 **SYNOPSIS**

12458 #include <strings.h>

12459 **DESCRIPTION**12460 The following shall be declared as functions and may also be defined as macros. Function
12461 prototypes shall be provided for use with ISO C standard compilers.

```

12462 XSI int ffs(int);
12463 int strcasecmp(const char *, const char *);
12464 int strcasecmp_l(const char *, const char *, locale_t);
12465 int strncasecmp(const char *, const char *, size_t);
12466 int strncasecmp_l(const char *, const char *, size_t, locale_t);

```

12467 The <strings.h> header shall define the **locale_t** type as described in <locale.h>.12468 The <strings.h> header shall define the **size_t** type as described in <sys/types.h>.12469 **APPLICATION USAGE**

12470 None.

12471 **RATIONALE**

12472 None.

12473 **FUTURE DIRECTIONS**

12474 None.

12475 **SEE ALSO**

12476 <locale.h>, <sys/types.h>

12477 XSH *ffs()*, *strcasecmp()*12478 **CHANGE HISTORY**

12479 First released in Issue 4, Version 2.

12480 **Issue 6**12481 The Open Group Corrigendum U021/2 is applied, correcting the prototype for *index()* to be
12482 consistent with the reference page.12483 The *bcmp()*, *bcopy()*, *bzero()*, *index()*, and *rindex()* functions are marked LEGACY.12484 **Issue 7**12485 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

12486 The LEGACY functions are removed.

12487 The <strings.h> header is moved from the XSI option to the Base.

12488 The *strcasecmp_l()* and *strncasecmp_l()* functions are added from The Open Group Technical
12489 Standard, 2006, Extended API Set Part 4.12490 A declaration for the **locale_t** type is added.

12491 **NAME**12492 stropts.h — STREAMS interface (**STREAMS**)12493 **SYNOPSIS**12494 OB XSR `#include <stropts.h>`12495 **DESCRIPTION**12496 The **<stropts.h>** header shall define the **bandinfo** structure, which shall include at least the
12497 following members:12498 int bi_flag Flushing type.
12499 unsigned char bi_pri Priority band.12500 The **<stropts.h>** header shall define the **strpeek** structure, which shall include at least the
12501 following members:12502 struct strbuf ctlbuf The control portion of the message.
12503 struct strbuf databuf The data portion of the message.
12504 t_uscalar_t flags RS_HIPRI or 0.12505 The **<stropts.h>** header shall define the **strbuf** structure, which shall include at least the
12506 following members:12507 char *buf Pointer to buffer.
12508 int len Length of data.
12509 int maxlen Maximum buffer length.12510 The **<stropts.h>** header shall define the **strfdinsert** structure, which shall include at least the
12511 following members:12512 struct strbuf ctlbuf The control portion of the message.
12513 struct strbuf databuf The data portion of the message.
12514 int fildes File descriptor of the other STREAM.
12515 t_uscalar_t flags RS_HIPRI or 0.
12516 int offset Relative location of the stored value.12517 The **<stropts.h>** header shall define the **striocctl** structure, which shall include at least the
12518 following members:12519 int ic_cmd *ioctl()* command.
12520 char *ic_dp Pointer to buffer.
12521 int ic_len Length of data.
12522 int ic_timeout Timeout for response.12523 The **<stropts.h>** header shall define the **strrecvfd** structure, which shall include at least the
12524 following members:12525 int fd Received file descriptor.
12526 gid_t gid GID of sender.
12527 uid_t uid UID of sender.12528 The **<stropts.h>** header shall define the **uid_t** and **gid_t** types through **typedef**, as described in
12529 **<sys/types.h>**.12530 The **<stropts.h>** header shall define the **t_scalar_t** and **t_uscalar_t** types, respectively, as signed
12531 and unsigned opaque types of equal length of at least 32 bits.12532 The **<stropts.h>** header shall define the **str_list** structure, which shall include at least the
12533 following members:

```

12534     struct str_mlist  *sl_modlist  STREAMS module names.
12535     int                sl_nmods    Number of STREAMS module names.

12536     The <stropts.h> header shall define the str_mlist structure, which shall include at least the
12537     following member:

12538     char  l_name[FMNAMESZ+1]  A STREAMS module name.

12539     The <stropts.h> header shall define at least the following symbolic constants for use as the
12540     request argument to ioctl():

12541     L_ATMARK      Is the top message ``marked''?
12542     L_CANPUT      Is a band writable?
12543     L_CKBAND      See if any messages exist in a band.
12544     L_FDINSERT    Send implementation-defined information about another STREAM.
12545     L_FIND        Look for a STREAMS module.
12546     L_FLUSH       Flush a STREAM.
12547     L_FLUSHBAND   Flush one band of a STREAM.
12548     L_GETBAND     Get the band of the top message on a STREAM.
12549     L_GETCLTIME   Get close time delay.
12550     L_GETSIG      Retrieve current notification signals.
12551     L_GRDOPT      Get the read mode.
12552     L_GWROPT      Get the write mode.
12553     L_LINK        Connect two STREAMS.
12554     L_LIST        Get all the module names on a STREAM.
12555     L_LOOK        Get the top module name.
12556     L_NREAD       Size the top message.
12557     L_PEEK        Peek at the top message on a STREAM.
12558     L_PLINK       Persistently connect two STREAMS.
12559     L_POP         Pop a STREAMS module.
12560     L_PUNLINK     Dismantle a persistent STREAMS link.
12561     L_PUSH        Push a STREAMS module.
12562     L_RECVFD      Get a file descriptor sent via L_SENDFD.
12563     L_SENDFD      Pass a file descriptor through a STREAMS pipe.
12564     L_SETCLTIME   Set close time delay.
12565     L_SETSIG      Ask for notification signals.
12566     L_SRDOPT      Set the read mode.
12567     L_STR         Send a STREAMS ioctl().

```

12568	L_SWROPT	Set the write mode.
12569	L_UNLINK	Disconnect two STREAMs.
12570	The <stropts.h> header shall define at least the following symbolic constant for use with	
12571	L_LOOK:	
12572	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.
12573	The <stropts.h> header shall define at least the following symbolic constants for use with	
12574	L_FLUSH:	
12575	FLUSHR	Flush read queues.
12576	FLUSHRW	Flush read and write queues.
12577	FLUSHW	Flush write queues.
12578	The <stropts.h> header shall define at least the following symbolic constants for use with	
12579	L_SETSIG:	
12580	S_BANDURG	When used in conjunction with S_RDBAND , SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
12581		
12582		
12583	S_ERROR	Notification of an error condition reaches the STREAM head.
12584	S_HANGUP	Notification of a hangup reaches the STREAM head.
12585	S_HIPRI	A high-priority message is present on a STREAM head read queue.
12586	S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue.
12587		
12588	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the front of the STREAM head read queue.
12589		
12590	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
12591		
12592		
12593	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue.
12594		
12595	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue.
12596		
12597	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full.
12598		
12599	S_WRNORM	Equivalent to S_OUTPUT .
12600	The <stropts.h> header shall define at least the following symbolic constant for use with	
12601	L_PEEK:	
12602	RS_HIPRI	Only look for high-priority messages.
12603	The <stropts.h> header shall define at least the following symbolic constants for use with	
12604	L_SRDOPT:	
12605	RMSGD	Message-discard mode.

12606	RMSGN	Message-non-discard mode.
12607	RNORM	Byte-STREAM mode, the default.
12608	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12609	RPROTDIS	Discard the control part of a message, delivering any data part, when a process issues a <i>read()</i> .
12610		
12611	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
12612		
12613	The <stropts.h> header shall define at least the following symbolic constant for use with L_SWOPT:	
12614		
12615	SNDZERO	Send a zero-length message downstream when a <i>write()</i> of 0 bytes occurs.
12616	The <stropts.h> header shall define at least the following symbolic constants for use with L_ATMARK:	
12617		
12618	ANYMARK	Check if the message is marked.
12619	LASTMARK	Check if the message is the last one marked on the queue.
12620	The <stropts.h> header shall define at least the following symbolic constant for use with L_UNLINK:	
12621		
12622	MUXID_ALL	Unlink all STREAMs linked to the STREAM associated with <i>files</i> .
12623	The <stropts.h> header shall define the following symbolic constants for <i>getmsg()</i> , <i>getpmsg()</i> , <i>putmsg()</i> , and <i>putpmsg()</i> :	
12624		
12625	MORECTL	More control information is left in message.
12626	MOREDATA	More data is left in message.
12627	MSG_ANY	Receive any message.
12628	MSG_BAND	Receive message from specified band.
12629	MSG_HIPRI	Send/receive high-priority message.
12630	The <stropts.h> header may make visible all of the symbols from <unistd.h>.	
12631	The <stropts.h> header may also define macros for message types using names that start with M_.	
12632		
12633	The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.	
12634		
12635	int	<code>fattach(int, const char *);</code>
12636	int	<code>fdetach(const char *);</code>
12637	int	<code>getmsg(int, struct strbuf *restrict, struct strbuf *restrict,</code>
12638		<code>int *restrict);</code>
12639	int	<code>getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,</code>
12640		<code>int *restrict, int *restrict);</code>
12641	int	<code>ioctl(int, int, ...);</code>
12642	int	<code>isastream(int);</code>
12643	int	<code>putmsg(int, const struct strbuf *, const struct strbuf *, int);</code>
12644	int	<code>putpmsg(int, const struct strbuf *, const struct strbuf *, int,</code>
12645		<code>int);</code>

12646 **APPLICATION USAGE**

12647 None.

12648 **RATIONALE**

12649 None.

12650 **FUTURE DIRECTIONS**

12651 None.

12652 **SEE ALSO**12653 [<sys/types.h>](#), [<unistd.h>](#)12654 XSH *close()*, *fattach()*, *fcntl()*, *fdetach()*, *getmsg()*, *ioctl()*, *isastream()*, *open()*, *pipe()*, *read()*, *poll()*,
12655 *putmsg()*, *signal()*, *write()*12656 **CHANGE HISTORY**

12657 First released in Issue 4, Version 2.

12658 **Issue 5**12659 The *flags* members of the **strpeek** and **strfdinsert** structures are changed from **type long** to
12660 **t_uscalar_t**.12661 **Issue 6**

12662 This header is marked as part of the XSI STREAMS Option Group.

12663 The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.12664 **Issue 7**12665 SD5-XBD-ERN-87 is applied, correcting an error in the **strrecvfd** structure.12666 The **<stropts.h>** header is marked obsolescent.

12667 This reference page is clarified with respect to macros and symbolic constants.

12668 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0076 [801] is applied.

12669 **NAME**

12670 sys/ipc.h — XSI interprocess communication access structure

12671 **SYNOPSIS**

12672 XSI `#include <sys/ipc.h>`

12673 **DESCRIPTION**

12674 The <sys/ipc.h> header is used by three mechanisms for XSI interprocess communication (IPC):
 12675 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm**, to pass
 12676 information used in determining permission to perform an IPC operation.

12677 The <sys/ipc.h> header shall define the **ipc_perm** structure, which shall include the following
 12678 members:

12679	uid_t	uid	Owner's user ID.
12680	gid_t	gid	Owner's group ID.
12681	uid_t	cuid	Creator's user ID.
12682	gid_t	cgid	Creator's group ID.
12683	mode_t	mode	Read/write permission.

12684 The <sys/ipc.h> header shall define the **uid_t**, **gid_t**, **mode_t**, and **key_t** types as described in
 12685 <sys/types.h>.

12686 The <sys/ipc.h> header shall define the following symbolic constants.

12687 Mode bits:

12688	IPC_CREAT	Create entry if key does not exist.
12689	IPC_EXCL	Fail if key exists.
12690	IPC_NOWAIT	Error if request must wait.

12691 Keys:

12692	IPC_PRIVATE	Private key.
-------	-------------	--------------

12693 Control commands:

12694	IPC_RMID	Remove identifier.
12695	IPC_SET	Set options.
12696	IPC_STAT	Get options.

12697 The following shall be declared as a function and may also be defined as a macro. A function
 12698 prototype shall be provided.

12699 `key_t ftok(const char *, int);`

12700 **APPLICATION USAGE**

12701 None.

12702 **RATIONALE**

12703 None.

12704 **FUTURE DIRECTIONS**

12705 None.

12706 **SEE ALSO**

12707 [<sys/types.h>](#)

12708 XSH *ftok()*

12709 **CHANGE HISTORY**

12710 First released in Issue 2. Derived from System V Release 2.0.

12711 **Issue 7**

12712 This reference page is clarified with respect to macros and symbolic constants.

12713 **NAME**

12714 sys/mman.h ‡memory management declarations

12715 **SYNOPSIS**

12716 #include <sys/mman.h>

12717 **DESCRIPTION**

12718 The <sys/mman.h> header shall define the following symbolic constants for use as protection
12719 options:

- 12720 PROT_EXEC Page can be executed.
- 12721 PROT_NONE Page cannot be accessed.
- 12722 PROT_READ Page can be read.
- 12723 PROT_WRITE Page can be written.

12724 The <sys/mman.h> header shall define the following symbolic constants for use as flag options:

- 12725 MAP_FIXED Interpret *addr* exactly.
- 12726 MAP_PRIVATE Changes are private.
- 12727 MAP_SHARED Share changes.

12728 XSI|SIO The <sys/mman.h> header shall define the following symbolic constants for the *msync()*
12729 function:

- 12730 MS_ASYNC Perform asynchronous writes.
- 12731 MS_INVALIDATE Invalidate mappings.
- 12732 MS_SYNC Perform synchronous writes.

12733 ML The <sys/mman.h> header shall define the following symbolic constants for the *mlockall()*
12734 function:

- 12735 MCL_CURRENT Lock currently mapped pages.
- 12736 MCL_FUTURE Lock pages that become mapped.

12737 The <sys/mman.h> header shall define the symbolic constant MAP_FAILED which shall have
12738 type **void *** and shall be used to indicate a failure from the *mmap()* function .

12739 ADV If the Advisory Information option is supported, the <sys/mman.h> header shall define
12740 symbolic constants for the *advice* argument to the *posix_madvise()* function as follows:

- 12741 POSIX_MADV_DONTNEED
12742 The application expects that it will not access the specified range in the near future.
- 12743 POSIX_MADV_NORMAL
12744 The application has no advice to give on its behavior with respect to the specified range. It
12745 is the default characteristic if no advice is given for a range of memory.
- 12746 POSIX_MADV_RANDOM
12747 The application expects to access the specified range in a random order.
- 12748 POSIX_MADV_SEQUENTIAL
12749 The application expects to access the specified range sequentially from lower addresses to
12750 higher addresses.

12751		POSIX_MADV_WILLNEED
12752		The application expects to access the specified range in the near future.
12753	TYM	The <sys/mman.h> header shall define the following symbolic constants for use as flags for the <i>posix_typed_mem_open()</i> function:
12754		
12755		POSIX_TYPED_MEM_ALLOCATE
12756		Allocate on <i>mmap()</i> .
12757		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12758		Allocate contiguously on <i>mmap()</i> .
12759		POSIX_TYPED_MEM_MAP_ALLOCATABLE
12760		Map on <i>mmap()</i> , without affecting allocatability.
12761		The <sys/mman.h> header shall define the mode_t , off_t , and size_t types as described in <sys/types.h>.
12762		
12763	TYM	The <sys/mman.h> header shall define the posix_typed_mem_info structure, which shall include at least the following member:
12764		
12765		<code>size_t posix_tmi_length</code> Maximum length which may be allocated
12766		from a typed memory object.
12767		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.
12768		
12769	MLR	<code>int mlock(const void *, size_t);</code>
12770	ML	<code>int mlockall(int);</code>
12771		<code>void *mmap(void *, size_t, int, int, int, off_t);</code>
12772		<code>int mprotect(void *, size_t, int);</code>
12773	XSI SIO	<code>int msync(void *, size_t, int);</code>
12774	MLR	<code>int munlock(const void *, size_t);</code>
12775	ML	<code>int munlockall(void);</code>
12776		<code>int munmap(void *, size_t);</code>
12777	ADV	<code>int posix_madvise(void *, size_t, int);</code>
12778	TYM	<code>int posix_mem_offset(const void *restrict, size_t, off_t *restrict,</code>
12779		<code>size_t *restrict, int *restrict);</code>
12780		<code>int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);</code>
12781		<code>int posix_typed_mem_open(const char *, int, int);</code>
12782	SHM	<code>int shm_open(const char *, int, mode_t);</code>
12783		<code>int shm_unlink(const char *);</code>

12784 **APPLICATION USAGE**

12785 None.

12786 **RATIONALE**

12787 None.

12788 **FUTURE DIRECTIONS**

12789 None.

12790 **SEE ALSO**

12791 <sys/types.h>

12792 XSH *mlock()*, *mlockall()*, *mmap()*, *mprotect()*, *msync()*, *munmap()*, *posix_madvise()*,
 12793 *posix_mem_offset()*, *posix_typed_mem_get_info()*, *posix_typed_mem_open()*, *shm_open()*,
 12794 *shm_unlink()*

12795 **CHANGE HISTORY**

12796 First released in Issue 4, Version 2.

12797 **Issue 5**

12798 Updated for alignment with the POSIX Realtime Extension.

12799 **Issue 6**

12800 The <sys/mman.h> header is marked as dependent on support for either the Memory Mapped
 12801 Files, Process Memory Locking, or Shared Memory Objects options.

12802 The following changes are made for alignment with IEEE Std 1003.1j-2000:

12803 The TYM margin code is added to the list of margin codes for the <sys/mman.h> header
 12804 line, as well as for other lines.

12805 The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG,
 12806 and POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.

12807 The **posix_tmi_length** structure is added.

12808 The *posix_mem_offset()*, *posix_typed_mem_get_info()*, and *posix_typed_mem_open()* functions
 12809 are added.

12810 The **restrict** keyword is added to the prototype for *posix_mem_offset()*.12811 IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for *posix_madvise()*.

12812 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and
 12813 shading errors for the *mlock()* and *munlock()* functions.

12814 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code
 12815 for the *mmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

12816 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code
 12817 for the *munmap()* function from MF|SHM to MC3 (notation for MF|SHM|TYM).

12818 **Issue 7**

12819 SD5-XBD-ERN-5 is applied, rewriting the DESCRIPTION.

12820 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
 12821 the Base.

12822 This reference page is clarified with respect to macros and symbolic constants.

12823 **NAME**

12824 sys/msg.h — XSI message queue structures

12825 **SYNOPSIS**

12826 XSI #include <sys/msg.h>

12827 **DESCRIPTION**

12828 The <sys/msg.h> header shall define the following data types through **typedef**:

12829 **msgqnum_t** Used for the number of messages in the message queue.

12830 **msglen_t** Used for the number of bytes allowed in a message queue.

12831 These types shall be unsigned integer types that are able to store values at least as large as a type
12832 **unsigned short**.

12833 The <sys/msg.h> header shall define the following symbolic constant as a message operation
12834 flag:

12835 MSG_NOERROR No error if big message.

12836 The <sys/msg.h> header shall define the **msqid_ds** structure, which shall include the following
12837 members:

12838	struct ipc_perm	msg_perm	Operation permission structure.
12839	msgqnum_t	msg_qnum	Number of messages currently on queue.
12840	msglen_t	msg_qbytes	Maximum number of bytes allowed on queue.
12841	pid_t	msg_lspid	Process ID of last <i>msgsnd()</i> .
12842	pid_t	msg_lrpid	Process ID of last <i>msgrcv()</i> .
12843	time_t	msg_stime	Time of last <i>msgsnd()</i> .
12844	time_t	msg_rtime	Time of last <i>msgrcv()</i> .
12845	time_t	msg_ctime	Time of last change.

12846 The <sys/msg.h> header shall define the **pid_t**, **size_t**, **ssize_t**, and **time_t** types as described in
12847 <sys/types.h>.

12848 The following shall be declared as functions and may also be defined as macros. Function
12849 prototypes shall be provided.

```

12850 int      msgctl(int, int, struct msqid_ds *);
12851 int      msgget(key_t, int);
12852 ssize_t  msgrcv(int, void *, size_t, long, int);
12853 int      msgsnd(int, const void *, size_t, int);

```

12854 In addition, the <sys/msg.h> header shall include the <sys/ipc.h> header.

12855 **APPLICATION USAGE**

12856 None.

12857 **RATIONALE**

12858 None.

12859 **FUTURE DIRECTIONS**

12860 None.

12861 **SEE ALSO**

12862 <sys/ipc.h>, <sys/types.h>

12863 XSH *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

12864 **CHANGE HISTORY**

12865 First released in Issue 2. Derived from System V Release 2.0.

12866 **Issue 7**

12867 Austin Group Interpretation 1003.1-2001 #179 is applied.

12868 This reference page is clarified with respect to macros and symbolic constants.

12869 **NAME**12870 `sys/resource.h` — definitions for XSI resource operations12871 **SYNOPSIS**12872 XSI `#include <sys/resource.h>`12873 **DESCRIPTION**12874 The **<sys/resource.h>** header shall define the following symbolic constants as possible values of
12875 the *which* argument of `getpriority()` and `setpriority()`:12876 `PRIO_PROCESS` Identifies the *who* argument as a process ID.12877 `PRIO_PGRP` Identifies the *who* argument as a process group ID.12878 `PRIO_USER` Identifies the *who* argument as a user ID.12879 The **<sys/resource.h>** header shall define the following type through **typedef**:12880 `rlim_t` Unsigned integer type used for limit values.12881 The **<sys/resource.h>** header shall define the following symbolic constants, which shall have
12882 values suitable for use in **#if** preprocessing directives:12883 `RLIM_INFINITY` A value of `rlim_t` indicating no limit.12884 `RLIM_SAVED_MAX` A value of type `rlim_t` indicating an unrepresentable saved hard
12885 limit.12886 `RLIM_SAVED_CUR` A value of type `rlim_t` indicating an unrepresentable saved soft limit.12887 On implementations where all resource limits are representable in an object of type `rlim_t`,
12888 `RLIM_SAVED_MAX` and `RLIM_SAVED_CUR` need not be distinct from `RLIM_INFINITY`.12889 The **<sys/resource.h>** header shall define the following symbolic constants as possible values of
12890 the *who* parameter of `getrusage()`:12891 `RUSAGE_SELF` Returns information about the current process.12892 `RUSAGE_CHILDREN` Returns information about children of the current process.12893 The **<sys/resource.h>** header shall define the `rlimit` structure, which shall include at least the
12894 following members:12895 `rlim_t rlim_cur` The current (soft) limit.12896 `rlim_t rlim_max` The hard limit.12897 The **<sys/resource.h>** header shall define the `rusage` structure, which shall include at least the
12898 following members:12899 `struct timeval ru_utime` User time used.12900 `struct timeval ru_stime` System time used.12901 The **<sys/resource.h>** header shall define the `timeval` structure as described in **<sys/time.h>**.12902 The **<sys/resource.h>** header shall define the following symbolic constants as possible values for
12903 the *resource* argument of `getrlimit()` and `setrlimit()`:12904 `RLIMIT_CORE` Limit on size of **core** file.12905 `RLIMIT_CPU` Limit on CPU time per process.

12906	RLIMIT_DATA	Limit on data segment size.
12907	RLIMIT_FSIZE	Limit on file size.
12908	RLIMIT_NOFILE	Limit on number of open files.
12909	RLIMIT_STACK	Limit on stack size.
12910	RLIMIT_AS	Limit on address space size.
12911	The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.	
12912		
12913	<code>int getpriority(int, id_t);</code>	
12914	<code>int getrlimit(int, struct rlimit *);</code>	
12915	<code>int getrusage(int, struct rusage *);</code>	
12916	<code>int setpriority(int, id_t, int);</code>	
12917	<code>int setrlimit(int, const struct rlimit *);</code>	
12918	The <sys/resource.h> header shall define the <code>id_t</code> type through <code>typedef</code> , as described in	
12919	<sys/types.h>.	
12920	Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.	
12921	APPLICATION USAGE	
12922	None.	
12923	RATIONALE	
12924	None.	
12925	FUTURE DIRECTIONS	
12926	None.	
12927	SEE ALSO	
12928	<sys/time.h> , <sys/types.h>	
12929	XSH getpriority() , getrlimit() , getrusage()	
12930	CHANGE HISTORY	
12931	First released in Issue 4, Version 2.	
12932	Issue 5	
12933	Large File System extensions are added.	
12934	Issue 7	
12935	This reference page is clarified with respect to macros and symbolic constants.	

12936 **NAME**

12937 sys/select.h ‡select types

12938 **SYNOPSIS**

12939 #include <sys/select.h>

12940 **DESCRIPTION**12941 The **<sys/select.h>** header shall define the **timeval** structure, which shall include at least the
12942 following members:

12943 time_t tv_sec Seconds.

12944 suseconds_t tv_usec Microseconds.

12945 The **<sys/select.h>** header shall define the **time_t** and **suseconds_t** types as described in
12946 **<sys/types.h>**.12947 The **<sys/select.h>** header shall define the **sigset_t** type as described in **<signal.h>**.12948 The **<sys/select.h>** header shall define the **timespec** structure as described in **<time.h>**.12949 The **<sys/select.h>** header shall define the **fd_set** type as a structure.12950 The **<sys/select.h>** header shall define the following symbolic constant, which shall have a value
12951 suitable for use in **#if** preprocessing directives:12952 FD_SETSIZE Maximum number of file descriptors in an **fd_set** structure.12953 The following shall be declared as functions, defined as macros, or both. If functions are
12954 declared, function prototypes shall be provided.

12955 void FD_CLR(int, fd_set *);

12956 int FD_ISSET(int, fd_set *);

12957 void FD_SET(int, fd_set *);

12958 void FD_ZERO(fd_set *);

12959 If implemented as macros, these may evaluate their arguments more than once, so applications
12960 should ensure that the arguments they supply are never expressions with side-effects.12961 The following shall be declared as functions and may also be defined as macros. Function
12962 prototypes shall be provided.12963 int pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12964 const struct timespec *restrict, const sigset_t *restrict);12965 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12966 struct timeval *restrict);12967 Inclusion of the **<sys/select.h>** header may make visible all symbols from the headers
12968 **<signal.h>** and **<time.h>**.12969 **APPLICATION USAGE**

12970 None.

12971 **RATIONALE**

12972 None.

12973 **FUTURE DIRECTIONS**

12974 None.

12975 **SEE ALSO**12976 [<signal.h>](#), [<sys/time.h>](#), [<sys/types.h>](#), [<time.h>](#)12977 XSH *pselect()*12978 **CHANGE HISTORY**

12979 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

12980 The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The
12981 Open Group Base Resolution bwg2001-005.12982 **Issue 7**

12983 SD5-XBD-ERN-6 is applied, reordering the DESCRIPTION.

12984 This reference page is clarified with respect to macros and symbolic constants.

12985 **NAME**

12986 sys/sem.h ‡XSI semaphore facility

12987 **SYNOPSIS**

12988 XSI #include <sys/sem.h>

12989 **DESCRIPTION**12990 The **<sys/sem.h>** header shall define the following symbolic constant for use as a semaphore
12991 operation flag:

12992 SEM_UNDO Set up adjust on exit entry.

12993 The **<sys/sem.h>** header shall define the following symbolic constants for use as commands for
12994 the *semctl()* function:12995 GETNCNT Get *semncnt*.12996 GETPID Get *sempid*.12997 GETVAL Get *semval*.12998 GETALL Get all cases of *semval*.12999 GETZCNT Get *semzcnt*.13000 SETVAL Set *semval*.13001 SETALL Set all cases of *semval*.13002 The **<sys/sem.h>** header shall define the **semid_ds** structure, which shall include the following
13003 members:

13004 struct ipc_perm sem_perm Operation permission structure.

13005 unsigned short sem_nsems Number of semaphores in set.

13006 time_t sem_otime Last *semop()* time.13007 time_t sem_ctime Last time changed by *semctl()*.13008 The **<sys/sem.h>** header shall define the **pid_t**, **size_t**, and **time_t** types as described in
13009 **<sys/types.h>**.13010 A semaphore shall be represented by an anonymous structure, which shall include the following
13011 members:

13012 unsigned short semval Semaphore value.

13013 pid_t sempid Process ID of last operation.

13014 unsigned short semncnt Number of processes waiting for *semval*
13015 to become greater than current value.13016 unsigned short semzcnt Number of processes waiting for *semval*
13017 to become 0.13018 The **<sys/sem.h>** header shall define the **sembuf** structure, which shall include the following
13019 members:

13020 unsigned short sem_num Semaphore number.

13021 short sem_op Semaphore operation.

13022 short sem_flg Operation flags.

13023 The following shall be declared as functions and may also be defined as macros. Function
13024 prototypes shall be provided.

13025 int semctl(int, int, int, ...);

```
13026     int    semget(key_t, int, int);
13027     int    semop(int, struct sembuf *, size_t);
```

13028 In addition, the <sys/sem.h> header shall include the <sys/ipc.h> header.

13029 APPLICATION USAGE

13030 None.

13031 RATIONALE

13032 None.

13033 FUTURE DIRECTIONS

13034 None.

13035 SEE ALSO

13036 <sys/ipc.h>, <sys/types.h>

13037 XSH *semctl()*, *semget()*, *semop()*

13038 CHANGE HISTORY

13039 First released in Issue 2. Derived from System V Release 2.0.

13040 Issue 7

13041 Austin Group Interpretation 1003.1-2001 #179 is applied.

13042 This reference page is clarified with respect to macros and symbolic constants.

13043 **NAME**

13044 sys/shm.h — XSI shared memory facility

13045 **SYNOPSIS**

13046 XSI #include <sys/shm.h>

13047 **DESCRIPTION**

13048 The <sys/shm.h> header shall define the following symbolic constants:

13049 SHM_RDONLY Attach read-only (else read-write).

13050 SHM_RND Round attach address to SHMLBA.

13051 SHMLBA Segment low boundary address multiple.

13052 The <sys/shm.h> header shall define the following data types through **typedef**:

13053 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
13054 store values at least as large as a type **unsigned short**.

13055 The <sys/shm.h> header shall define the **shmid_ds** structure, which shall include the following
13056 members:

13057	struct ipc_perm	shm_perm	Operation permission structure.
13058	size_t	shm_segsz	Size of segment in bytes.
13059	pid_t	shm_lpid	Process ID of last shared memory operation.
13060	pid_t	shm_cpid	Process ID of creator.
13061	shmatt_t	shm_nattch	Number of current attaches.
13062	time_t	shm_atime	Time of last <i>shmat</i> ().
13063	time_t	shm_dtime	Time of last <i>shmdt</i> ().
13064	time_t	shm_ctime	Time of last change by <i>shmctl</i> ().

13065 The <sys/shm.h> header shall define the **pid_t**, **size_t**, and **time_t** types as described in
13066 <sys/types.h>.

13067 The following shall be declared as functions and may also be defined as macros. Function
13068 prototypes shall be provided.

```

13069 void *shmat(int, const void *, int);
13070 int shmctl(int, int, struct shmid_ds *);
13071 int shmdt(const void *);
13072 int shmget(key_t, size_t, int);

```

13073 In addition, the <sys/shm.h> header shall include the <sys/ipc.h> header.

13074 **APPLICATION USAGE**

13075 None.

13076 **RATIONALE**

13077 None.

13078 **FUTURE DIRECTIONS**

13079 None.

13080 **SEE ALSO**

13081 <sys/ipc.h>, <sys/types.h>

13082 XSH *shmat*(), *shmctl*(), *shmdt*(), *shmget*()

13083 **CHANGE HISTORY**

13084 First released in Issue 2. Derived from System V Release 2.0.

13085 **Issue 5**

13086 The type of *shm_segsz* is changed from **int** to **size_t**.

13087 **Issue 7**

13088 Austin Group Interpretation 1003.1-2001 #179 is applied.

13089 This reference page is clarified with respect to macros and symbolic constants.

13090 **NAME**

13091 sys/socket.h ‡main sockets header

13092 **SYNOPSIS**

13093 #include <sys/socket.h>

13094 **DESCRIPTION**13095 The **<sys/socket.h>** header shall define the **socklen_t** type, which is an integer type of width of
13096 at least 32 bits; see APPLICATION USAGE.13097 The **<sys/socket.h>** header shall define the **sa_family_t** unsigned integer type.13098 The **<sys/socket.h>** header shall define the **sockaddr** structure, which shall include at least the
13099 following members:13100 sa_family_t sa_family Address family.
13101 char sa_data[] Socket address (variable-length data).13102 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
13103 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.13104 The **<sys/socket.h>** header shall define the **sockaddr_storage** structure, which shall be:13105 Large enough to accommodate all supported protocol-specific address structures
13106 Aligned at an appropriate boundary so that pointers to it can be cast as pointers to
13107 protocol-specific address structures and used to access the fields of those structures
13108 without alignment problems13109 The **sockaddr_storage** structure shall include at least the following members:

13110 sa_family_t ss_family

13111 When a pointer to a **sockaddr_storage** structure is cast as a pointer to a **sockaddr** structure, the
13112 *ss_family* field of the **sockaddr_storage** structure shall map onto the *sa_family* field of the
13113 **sockaddr** structure. When a pointer to a **sockaddr_storage** structure is cast as a pointer to a
13114 protocol-specific address structure, the *ss_family* field shall map onto a field of that structure that
13115 is of type **sa_family_t** and that identifies the protocol's address family.13116 The **<sys/socket.h>** header shall define the **msghdr** structure, which shall include at least the
13117 following members:13118 void *msg_name Optional address.
13119 socklen_t msg_namelen Size of address.
13120 struct iovec *msg_iov Scatter/gather array.
13121 int msg_iovlen Members in *msg_iov*.
13122 void *msg_control Ancillary data; see below.
13123 socklen_t msg_controllen Ancillary data buffer *len*.
13124 int msg_flags Flags on received message.13125 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
13126 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the
13127 *recvmsg()* function and *value* only for the *sendmsg()* function.13128 The **<sys/socket.h>** header shall define the **iovec** structure as described in **<sys/uio.h>**.13129 The **<sys/socket.h>** header shall define the **cmsghdr** structure, which shall include at least the
13130 following members:13131 socklen_t cmsg_len Data byte count, including the **cmsghdr**.
13132 int cmsg_level Originating protocol.

13133 int msg_type Protocol-specific type.

13134 The **msg_hdr** structure is used for storage of ancillary data object information.

13135 Ancillary data consists of a sequence of pairs, each consisting of a **msg_hdr** structure followed
 13136 by a data array. The data array contains the ancillary data message, and the **msg_hdr** structure
 13137 contains descriptive information that allows an application to correctly parse the data.

13138 The values for *msg_level* shall be legal values for the *level* argument to the *getsockopt()* and
 13139 *setsockopt()* functions. The system documentation shall specify the *msg_type* definitions for the
 13140 supported protocols.

13141 Ancillary data is also possible at the socket level. The <sys/socket.h> header shall define the
 13142 following symbolic constant for use as the *msg_type* value when *msg_level* is SOL_SOCKET:

13143 SCM_RIGHTS Indicates that the data array contains the access rights to be sent or
 13144 received.

13145 The <sys/socket.h> header shall define the following macros to gain access to the data arrays in
 13146 the ancillary data associated with a message header:

13147 MSG_DATA(*msg*)
 13148 If the argument is a pointer to a **msg_hdr** structure, this macro shall return an unsigned
 13149 character pointer to the data array associated with the **msg_hdr** structure.

13150 MSG_NXTHDR(*mhdr, msg*)
 13151 If the first argument is a pointer to a **msg_hdr** structure and the second argument is a pointer
 13152 to a **msg_hdr** structure in the ancillary data pointed to by the *msg_control* field of that
 13153 **msg_hdr** structure, this macro shall return a pointer to the next **msg_hdr** structure, or a null
 13154 pointer if this structure is the last **msg_hdr** in the ancillary data.

13155 MSG_FIRSTHDR(*mhdr*)
 13156 If the argument is a pointer to a **msg_hdr** structure, this macro shall return a pointer to the
 13157 first **msg_hdr** structure in the ancillary data associated with this **msg_hdr** structure, or a null
 13158 pointer if there is no ancillary data associated with the **msg_hdr** structure.

13159 The <sys/socket.h> header shall define the **linger** structure, which shall include at least the
 13160 following members:

13161 int l_onoff Indicates whether linger option is enabled.
 13162 int l_linger Linger time, in seconds.

13163 The <sys/socket.h> header shall define the following symbolic constants with distinct values:

13164 SOCK_DGRAM Datagram socket.

13165 RS SOCK_RAW Raw Protocol Interface.

13166 SOCK_SEQPACKET Sequenced-packet socket.

13167 SOCK_STREAM Byte-stream socket.

13168 The <sys/socket.h> header shall define the following symbolic constant for use as the *level*
 13169 argument of *setsockopt()* and *getsockopt()*.

13170 SOL_SOCKET Options to be accessed at socket level, not protocol level.

13171 The <sys/socket.h> header shall define the following symbolic constants with distinct values for
 13172 use as the *option_name* argument in *getsockopt()* or *setsockopt()* calls (see XSH Section 2.10.16, on
 13173 page 528):

13174	SO_ACCEPTCONN	Socket is accepting connections.
13175	SO_BROADCAST	Transmission of broadcast messages is supported.
13176	SO_DEBUG	Debugging information is being recorded.
13177	SO_DONTROUTE	Bypass normal routing.
13178	SO_ERROR	Socket error status.
13179	SO_KEEPALIVE	Connections are kept alive with periodic messages.
13180	SO_LINGER	Socket lingers on close.
13181	SO_OOBINLINE	Out-of-band data is transmitted in line.
13182	SO_RCVBUF	Receive buffer size.
13183	SO_RCVLOWAT	Receive ``low water mark``.
13184	SO_RCVTIMEO	Receive timeout.
13185	SO_REUSEADDR	Reuse of local addresses is supported.
13186	SO_SNDBUF	Send buffer size.
13187	SO_SNDLOWAT	Send ``low water mark``.
13188	SO_SNDTIMEO	Send timeout.
13189	SO_TYPE	Socket type.
13190	The <sys/socket.h> header shall define the following symbolic constant for use as the maximum	
13191	<i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
13192	SOMAXCONN	The maximum <i>backlog</i> queue length.
13193	The <sys/socket.h> header shall define the following symbolic constants with distinct values for	
13194	use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in	
13195	<i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
13196	MSG_CTRUNC	Control data truncated.
13197	MSG_DONTROUTE	Send without using routing tables.
13198	MSG_EOR	Terminates a record (if supported by the protocol).
13199	MSG_OOB	Out-of-band data.
13200	MSG_NOSIGNAL	No SIGPIPE generated when an attempt to send is made on a stream-
13201		oriented socket that is no longer connected.
13202	MSG_PEEK	Leave received data in queue.
13203	MSG_TRUNC	Normal data truncated.
13204	MSG_WAITALL	Attempt to fill the read buffer.
13205	The <sys/socket.h> header shall define the following symbolic constants with distinct values:	
13206	AF_INET	Internet domain sockets for use with IPv4 addresses.
13207	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.
13208	AF_UNIX	UNIX domain sockets.

13209 AF_UNSPEC Unspecified.

13210 The value of AF_UNSPEC shall be 0.

13211 The <sys/socket.h> header shall define the following symbolic constants with distinct values:

13212 SHUT_RD Disables further receive operations.

13213 SHUT_RDWR Disables further send and receive operations.

13214 SHUT_WR Disables further send operations.

13215 The <sys/socket.h> header shall define the `size_t` and `ssize_t` types as described in
13216 <sys/types.h>.

13217 The following shall be declared as functions and may also be defined as macros. Function
13218 prototypes shall be provided.

```
13219 int accept(int, struct sockaddr *restrict, socklen_t *restrict);
13220 int bind(int, const struct sockaddr *, socklen_t);
13221 int connect(int, const struct sockaddr *, socklen_t);
13222 int getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
13223 int getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
13224 int getsockopt(int, int, int, void *restrict, socklen_t *restrict);
13225 int listen(int, int);
13226 ssize_t recv(int, void *, size_t, int);
13227 ssize_t recvfrom(int, void *restrict, size_t, int,
13228 struct sockaddr *restrict, socklen_t *restrict);
13229 ssize_t recvmsg(int, struct msghdr *, int);
13230 ssize_t send(int, const void *, size_t, int);
13231 ssize_t sendmsg(int, const struct msghdr *, int);
13232 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
13233 socklen_t);
13234 int setsockopt(int, int, int, const void *, socklen_t);
13235 int shutdown(int, int);
13236 int socketatmark(int);
13237 int socket(int, int, int);
13238 int socketpair(int, int, int, int [2]);
```

13239 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/uiio.h>.

13240 APPLICATION USAGE

13241 To forestall portability problems, it is recommended that applications not use values larger than
13242 $2^{31} - 1$ for the `socklen_t` type.

13243 The `sockaddr_storage` structure solves the problem of declaring storage for automatic variables
13244 which is both large enough and aligned enough for storing the socket address data structure of
13245 any family. For example, code with a file descriptor and without the context of the address
13246 family can pass a pointer to a variable of this type, where a pointer to a socket address structure
13247 is expected in calls such as `getpeername()`, and determine the address family by accessing the
13248 received content after the call.

13249 The example below illustrates a data structure which aligns on a 64-bit boundary. An
13250 implementation-defined field `_ss_align` following `_ss_pad1` is used to force a 64-bit alignment
13251 which covers proper alignment good enough for needs of at least `sockaddr_in6` (IPv6) and
13252 `sockaddr_in` (IPv4) address data structures. The size of padding field `_ss_pad1` depends on the
13253 chosen alignment boundary. The size of padding field `_ss_pad2` depends on the value of overall
13254 size chosen for the total size of the structure. This size and alignment are represented in the

13255 above example by implementation-defined (not required) constants `_SS_MAXSIZE` (chosen
 13256 value 128) and `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived
 13257 value 6) and `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The
 13258 implementation-defined definitions and structure field names above start with an <underscore>
 13259 to denote implementation private name space. Portable code is not expected to access or
 13260 reference those fields or constants.

```

13261  /*
13262  *   Desired design of maximum size and alignment.
13263  */
13264  #define _SS_MAXSIZE 128
13265  /* Implementation-defined maximum size. */
13266  #define _SS_ALIGNSIZE (sizeof(int64_t))
13267  /* Implementation-defined desired alignment. */

13268  /*
13269  *   Definitions used for sockaddr_storage structure paddings design.
13270  */
13271  #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
13272  #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+ \
13273  _SS_PAD1SIZE + _SS_ALIGNSIZE))
13274  struct sockaddr_storage {
13275  sa_family_t  ss_family; /* Address family. */
13276  /*
13277  *   Following fields are implementation-defined.
13278  */
13279  char  _ss_pad1[_SS_PAD1SIZE];
13280  /* 6-byte pad; this is to make implementation-defined
13281  pad up to alignment field that follows explicit in
13282  the data structure. */
13283  int64_t  _ss_align; /* Field to force desired structure
13284  storage alignment. */
13285  char  _ss_pad2[_SS_PAD2SIZE];
13286  /* 112-byte pad to achieve desired size,
13287  _SS_MAXSIZE value minus size of ss_family
13288  __ss_pad1, __ss_align fields is 112. */
13289  };

```

13290 RATIONALE

13291 None.

13292 FUTURE DIRECTIONS

13293 None.

13294 SEE ALSO

13295 [<sys/types.h>](#), [<sys/uio.h>](#)

13296 *XSH* [accept\(\)](#), [bind\(\)](#), [connect\(\)](#), [getpeername\(\)](#), [getsockname\(\)](#), [getsockopt\(\)](#), [listen\(\)](#), [recv\(\)](#),
 13297 [recvfrom\(\)](#), [recvmsg\(\)](#), [send\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#), [shutdown\(\)](#), [socketatmark\(\)](#), [socket\(\)](#),
 13298 [socketpair\(\)](#)

13299 CHANGE HISTORY

13300 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13301 The **restrict** keyword is added to the prototypes for [accept\(\)](#), [getpeername\(\)](#), [getsockname\(\)](#),
 13302 [getsockopt\(\)](#), and [recvfrom\(\)](#).

13303 **Issue 7**

13304 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the `ssize_t` type.

13305 SD5-XBD-ERN-62 is applied.

13306 The `MSG_NOSIGNAL` symbolic constant is added from The Open Group Technical Standard,
13307 2006, Extended API Set Part 2.

13308 This reference page is clarified with respect to macros and symbolic constants, and a declaration
13309 for the `size_t` type is added.

13310 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0067 [355] is applied.

13311 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0077 [934] is applied.

13312 **NAME**

13313 sys/stat.h — data returned by the stat() function

13314 **SYNOPSIS**

13315 #include <sys/stat.h>

13316 **DESCRIPTION**

13317 The <sys/stat.h> header shall define the structure of the data returned by the *fstat()*, *lstat()*, and
13318 *stat()* functions.

13319 The <sys/stat.h> header shall define the **stat** structure, which shall include at least the following
13320 members:

13321	dev_t st_dev	Device ID of device containing file.
13322	ino_t st_ino	File serial number.
13323	mode_t st_mode	Mode of file (see below).
13324	nlink_t st_nlink	Number of hard links to the file.
13325	uid_t st_uid	User ID of file.
13326	gid_t st_gid	Group ID of file.
13327 XSI	dev_t st_rdev	Device ID (if file is character or block special).
13328	off_t st_size	For regular files, the file size in bytes.
13329		For symbolic links, the length in bytes of the
13330		pathname contained in the symbolic link.
13331 SHM		For a shared memory object, the length in bytes.
13332 TYM		For a typed memory object, the length in bytes.
13333		For other file types, the use of this field is
13334		unspecified.
13335	struct timespec st_atim	Last data access timestamp.
13336	struct timespec st_mtim	Last data modification timestamp.
13337	struct timespec st_ctim	Last file status change timestamp.
13338 XSI	blksize_t st_blksize	A file system-specific preferred I/O block size
13339		for this object. In some file system types, this
13340		may vary from file to file.
13341	blkcnt_t st_blocks	Number of blocks allocated for this object.

13342 The *st_ino* and *st_dev* fields taken together uniquely identify the file within the system.

13343 XSI The <sys/stat.h> header shall define the **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**,
13344 **uid_t**, **gid_t**, **off_t**, and **time_t** types as described in <sys/types.h>.

13345 The <sys/stat.h> header shall define the **timespec** structure as described in <time.h>. Times
13346 shall be given in seconds since the Epoch.

13347 Which structure members have meaningful values depends on the type of file. For further
13348 information, see the descriptions of *fstat()*, *lstat()*, and *stat()* in the System Interfaces volume of
13349 POSIX.1-2017.

13350 For compatibility with earlier versions of this standard, the *st_atime* macro shall be defined with
13351 the value *st_atim.tv_sec*. Similarly, *st_ctime* and *st_mtime* shall be defined as macros with the
13352 values *st_ctim.tv_sec* and *st_mtim.tv_sec*, respectively.

13353 The <sys/stat.h> header shall define the following symbolic constants for the file types encoded
 13354 in type **mode_t**. The values shall be suitable for use in **#if** preprocessing directives:

13355	XSI	S_IFMT	Type of file.
13356		S_IFBLK	Block special.
13357		S_IFCHR	Character special.
13358		S_IFIFO	FIFO special.
13359		S_IFREG	Regular.
13360		S_IFDIR	Directory.
13361		S_IFLNK	Symbolic link.
13362		S_IFSOCK	Socket.

13363 The <sys/stat.h> header shall define the following symbolic constants for the file mode bits
 13364 encoded in type **mode_t**, with the indicated numeric values. These macros shall expand to an
 13365 expression which has a type that allows them to be used, either singly or OR'ed together, as the
 13366 third argument to *open()* without the need for a **mode_t** cast. The values shall be suitable for use
 13367 in **#if** preprocessing directives.

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
XSI S_ISVTX	01000	On directories, restricted deletion flag.

13384 The following macros shall be provided to test whether a file is of the specified type. The value
 13385 *m* supplied to the macros is the value of *st_mode* from a **stat** structure. The macro shall evaluate
 13386 to a non-zero value if the test is true; 0 if the test is false.

- 13387 **S_ISBLK**(*m*) Test for a block special file.
- 13388 **S_ISCHR**(*m*) Test for a character special file.
- 13389 **S_ISDIR**(*m*) Test for a directory.
- 13390 **S_ISFIFO**(*m*) Test for a pipe or FIFO special file.
- 13391 **S_ISREG**(*m*) Test for a regular file.

13392 S_ISLNK(*m*) Test for a symbolic link.

13393 S_ISSOCK(*m*) Test for a socket.

13394 The implementation may implement message queues, semaphores, or shared memory objects as
13395 distinct file types. The following macros shall be provided to test whether a file is of the
13396 specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat**
13397 structure. The macro shall evaluate to a non-zero value if the specified object is implemented as
13398 a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*.
13399 Otherwise, the macro shall evaluate to zero.

13400 S_TYPEISMQ(*buf*) Test for a message queue.

13401 S_TYPEISSEM(*buf*) Test for a semaphore.

13402 S_TYPEISSHM(*buf*) Test for a shared memory object.

13403 TYM The implementation may implement typed memory objects as distinct file types, and the
13404 following macro shall test whether a file is of the specified type. The value of the *buf* argument
13405 supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero
13406 value if the specified object is implemented as a distinct file type and the specified file type is
13407 contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

13408 S_TYPEISTMO(*buf*) Test macro for a typed memory object.

13409 The <sys/stat.h> header shall define the following symbolic constants as distinct integer values
13410 outside of the range [0,999 999 999], for use with the *futimens()* and *utimensat()* functions:

13411 UTIME_NOW

13412 UTIME_OMIT

13413 The following shall be declared as functions and may also be defined as macros. Function
13414 prototypes shall be provided.

13415 int chmod(const char *, mode_t);

13416 int fchmod(int, mode_t);

13417 int fchmodat(int, const char *, mode_t, int);

13418 int fstat(int, struct stat *);

13419 int fstatat(int, const char *restrict, struct stat *restrict, int);

13420 int futimens(int, const struct timespec [2]);

13421 int lstat(const char *restrict, struct stat *restrict);

13422 int mkdir(const char *, mode_t);

13423 int mkdirat(int, const char *, mode_t);

13424 int mkfifo(const char *, mode_t);

13425 int mkfifoat(int, const char *, mode_t);

13426 XSI int mknod(const char *, mode_t, dev_t);

13427 int mknodat(int, const char *, mode_t, dev_t);

13428 int stat(const char *restrict, struct stat *restrict);

13429 mode_t umask(mode_t);

13430 int utimensat(int, const char *, const struct timespec [2], int);

13431 Inclusion of the <sys/stat.h> header may make visible all symbols from the <time.h> header.

13432 **APPLICATION USAGE**

13433 Use of the macros is recommended for determining the type of a file.

13434 **RATIONALE**

13435 A conforming C-language application must include <sys/stat.h> for functions that have
13436 arguments or return values of type **mode_t**, so that symbolic values for that type can be used.
13437 An alternative would be to require that these constants are also defined by including
13438 <sys/types.h>.

13439 The S_ISUID and S_ISGID bits may be cleared on any write, not just on *open()*, as some
13440 historical implementations do.

13441 System calls that update the time entry fields in the **stat** structure must be documented by the
13442 implementors. POSIX-conforming systems should not update the time entry fields for functions
13443 listed in the System Interfaces volume of POSIX.1-2017 unless the standard requires that they do,
13444 except in the case of documented extensions to the standard.

13445 Upon assignment, file timestamps are immediately converted to the resolution of the file system
13446 by truncation (i.e., the recorded time can be older than the actual time). For example, if the file
13447 system resolution is 1 microsecond, then a conforming *stat()* must always return an
13448 *st_mtim.tv_nsec* that is a multiple of 1000. Some older implementations returned higher-
13449 resolution timestamps while the *inode* information was cached, and then spontaneously
13450 truncated the *tv_nsec* fields when they were stored to and retrieved from disk, but this behavior
13451 does not conform.

13452 Note that *st_dev* must be unique within a Local Area Network (LAN) in a ``system'' made up of
13453 multiple computers' file systems connected by a LAN.

13454 Networked implementations of a POSIX-conforming system must guarantee that all files visible
13455 within the file tree (including parts of the tree that may be remotely mounted from other
13456 machines on the network) on each individual processor are uniquely identified by the
13457 combination of the *st_ino* and *st_dev* fields.

13458 The unit for the *st_blocks* member of the **stat** structure is not defined within POSIX.1-2017. In
13459 some implementations it is 512 bytes. It may differ on a file system basis. There is no correlation
13460 between values of the *st_blocks* and *st_blksize*, and the *f_bsize* (from <sys/statvfs.h>) structure
13461 members.

13462 Traditionally, some implementations defined the multiplier for *st_blocks* in <sys/param.h> as the
13463 symbol DEV_BSIZE.

13464 Some earlier versions of this standard did not specify values for the file mode bit macros. The
13465 expectation was that some implementors might choose to use a different encoding for these bits
13466 than the traditional one, and that new applications would use symbolic file modes instead of
13467 numeric. This version of the standard specifies the traditional encoding, in recognition that
13468 nearly 20 years after the first publication of this standard numeric file modes are still in
13469 widespread use by application developers, and that all conforming implementations still use the
13470 traditional encoding.

13471 **FUTURE DIRECTIONS**

13472 No new S_IFMT symbolic names for the file type values of **mode_t** will be defined by
13473 POSIX.1-2017; if new file types are required, they will only be testable through *S_ISxx()* or
13474 *S_TYPEISxxx()* macros instead.

13475 **SEE ALSO**13476 [<sys/statvfs.h>](#), [<sys/types.h>](#), [<time.h>](#)13477 XSH *chmod()*, *fchmod()*, *fstat()*, *fstatat()*, *futimens()*, *mkdir()*, *mkfifo()*, *mknod()*, *umask()*13478 **CHANGE HISTORY**

13479 First released in Issue 1. Derived from Issue 1 of the SVID.

13480 **Issue 5**

13481 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

13482 The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from
13483 **long** to **blkcnt_t**.13484 **Issue 6**13485 The S_TYPEISMQ(), S_TYPEISSEM(), and S_TYPEISSHM() macros are unconditionally
13486 mandated.13487 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t**
13488 and **blkcnt_t** have been described.13489 The following new requirements on POSIX implementations derive from alignment with the
13490 Single UNIX Specification:13491 The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

13492 S_IFSOCK and S_ISSOCK are added for sockets.

13493 The description of **stat** structure members is changed to reflect contents when file type is a
13494 symbolic link.

13495 The test macro S_TYPEISTMO is added for alignment with IEEE Std 1003.1j-2000.

13496 The **restrict** keyword is added to the prototypes for *lstat()* and *stat()*.13497 The *lstat()* function is made mandatory.13498 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the
13499 *st_blocks* member of the **stat** structure to the RATIONALE.13500 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the
13501 DESCRIPTION that the **timespec** structure may be defined as described in the **<time.h>** header.13502 **Issue 7**13503 SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the
13504 interfaces should be consulted in order to determine which structure members have meaningful
13505 values.13506 The *fchmodat()*, *fstatat()*, *mkdirat()*, *mkfifoat()*, *mknodat()*, and *utimensat()* functions are added
13507 from The Open Group Technical Standard, 2006, Extended API Set Part 2.13508 The *futimens()* function is added.

13509 This reference page is clarified with respect to macros and symbolic constants.

13510 Changes are made related to support for finegrained timestamps and the **UTIME_NOW** and
13511 **UTIME_OMIT** symbolic constants are added.

13512 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0068 [207] is applied.

13513 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0078 [531] is applied.

13514 **NAME**

13515 sys/statvfs.h — VFS File System information structure

13516 **SYNOPSIS**

13517 #include <sys/statvfs.h>

13518 **DESCRIPTION**

13519 The <sys/statvfs.h> header shall define the **statvfs** structure, which shall include at least the
 13520 following members:

13521	unsigned long	f_bsize	File system block size.
13522	unsigned long	f_frsize	Fundamental file system block size.
13523	fsblkcnt_t	f_blocks	Total number of blocks on file system in units of <i>f_frsize</i> .
13524	fsblkcnt_t	f_bfree	Total number of free blocks.
13525	fsblkcnt_t	f_bavail	Number of free blocks available to non-privileged process.
13526			
13527	fsfilcnt_t	f_files	Total number of file serial numbers.
13528	fsfilcnt_t	f_ffree	Total number of free file serial numbers.
13529	fsfilcnt_t	f_favail	Number of file serial numbers available to non-privileged process.
13530			
13531	unsigned long	f_fsid	File system ID.
13532	unsigned long	f_flag	Bit mask of <i>f_flag</i> values.
13533	unsigned long	f_namemax	Maximum filename length.

13534 The <sys/statvfs.h> header shall define the **fsblkcnt_t** and **fsfilcnt_t** types as described in
 13535 <sys/types.h>.

13536 The <sys/statvfs.h> header shall define the following symbolic constants for the *f_flag* member:

13537	ST_RDONLY	Read-only file system.
13538	ST_NOSUID	Does not support the semantics of the ST_ISUID and ST_ISGID file mode bits.

13539 The following shall be declared as functions and may also be defined as macros. Function
 13540 prototypes shall be provided.

```
13541 int fstatvfs(int, struct statvfs *);
13542 int statvfs(const char *restrict, struct statvfs *restrict);
```

13543 **APPLICATION USAGE**

13544 None.

13545 **RATIONALE**

13546 None.

13547 **FUTURE DIRECTIONS**

13548 None.

13549 **SEE ALSO**

13550 <sys/types.h>

13551 XSH *fstatvfs()*

13552 **CHANGE HISTORY**

13553 First released in Issue 4, Version 2.

13554 **Issue 5**

13555 The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type of
 13556 *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

13557 **Issue 6**

13558 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t**
13559 and **fsfilcnt_t** have been described.

13560 The **restrict** keyword is added to the prototype for *statvfs()*.

13561 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of
13562 ST_NOSUID from “Does not support *setuid()/setgid()* semantics” to “Does not support the
13563 semantics of the ST_ISUID and ST_ISGID file mode bits”.

13564 **Issue 7**

13565 The **<sys/statvfs.h>** header is moved from the XSI option to the Base.

13566 This reference page is clarified with respect to macros and symbolic constants.

13567 **NAME**

13568 sys/time.h ‡time types

13569 **SYNOPSIS**

13570 XSI #include <sys/time.h>

13571 **DESCRIPTION**

13572 The <sys/time.h> header shall define the **timeval** structure, which shall include at least the
13573 following members:

13574 time_t tv_sec Seconds.
13575 suseconds_t tv_usec Microseconds.

13576 OB The <sys/time.h> header shall define the **itimerval** structure, which shall include at least the
13577 following members:

13578 struct timeval it_interval Timer interval.
13579 struct timeval it_value Current value.

13580 The <sys/time.h> header shall define the **time_t** and **suseconds_t** types as described in
13581 <sys/types.h>.

13582 The <sys/time.h> header shall define the **fd_set** type as described in <sys/select.h>.

13583 OB The <sys/time.h> header shall define the following symbolic constants for the *which* argument of
13584 *getitimer()* and *setitimer()*:

13585 ITIMER_REAL Decrements in real time.
13586 ITIMER_VIRTUAL Decrements in process virtual time.
13587 ITIMER_PROF Decrements both in process virtual time and when the system is running
13588 on behalf of the process.

13589 The <sys/time.h> header shall define the following as described in <sys/select.h>:

13590 FD_CLR()
13591 FD_ISSET()
13592 FD_SET()
13593 FD_ZERO()
13594 FD_SETSIZE

13595 The following shall be declared as functions and may also be defined as macros. Function
13596 prototypes shall be provided.

13597 OB int getitimer(int, struct itimerval *);
13598 int gettimeofday(struct timeval *restrict, void *restrict);
13599 int setitimer(int, const struct itimerval *restrict,
13600 struct itimerval *restrict);
13601 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
13602 struct timeval *restrict);
13603 int utimes(const char *, const struct timeval [2]);

13604 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>
13605 header.

13606 **APPLICATION USAGE**

13607 None.

13608 **RATIONALE**

13609 None.

13610 **FUTURE DIRECTIONS**

13611 None.

13612 **SEE ALSO**

13613 [<sys/select.h>](#), [<sys/types.h>](#)

13614 XSH *futimens()*, *getitimer()*, *gettimeofday()*, *pselect()*

13615 **CHANGE HISTORY**

13616 First released in Issue 4, Version 2.

13617 **Issue 5**

13618 The type of *tv_usec* is changed from **long** to **suseconds_t**.

13619 **Issue 6**

13620 The **restrict** keyword is added to the prototypes for *gettimeofday()*, *select()*, and *setitimer()*.

13621 The note is added that inclusion of this header may also make symbols visible from
13622 [<sys/select.h>](#).

13623 The *utimes()* function is marked LEGACY.

13624 **Issue 7**

13625 This reference page is clarified with respect to macros and symbolic constants.

13626 **NAME**

13627 sys/times.h — file access and modification times structure

13628 **SYNOPSIS**

13629 #include <sys/times.h>

13630 **DESCRIPTION**13631 The <sys/times.h> header shall define the **tms** structure, which is returned by *times()* and shall
13632 include at least the following members:

13633 clock_t tms_utime User CPU time.

13634 clock_t tms_stime System CPU time.

13635 clock_t tms_cutime User CPU time of terminated child processes.

13636 clock_t tms_cstime System CPU time of terminated child processes.

13637 The <sys/times.h> header shall define the **clock_t** type as described in <sys/types.h>.13638 The following shall be declared as a function and may also be defined as a macro. A function
13639 prototype shall be provided.

13640 clock_t times(struct tms *);

13641 **APPLICATION USAGE**

13642 None.

13643 **RATIONALE**

13644 None.

13645 **FUTURE DIRECTIONS**

13646 None.

13647 **SEE ALSO**

13648 <sys/types.h>

13649 XSH *times()*13650 **CHANGE HISTORY**

13651 First released in Issue 1. Derived from Issue 1 of the SVID.

13652 **NAME**

13653 sys/types.h ¶data types

13654 **SYNOPSIS**

13655 #include <sys/types.h>

13656 **DESCRIPTION**

13657 The <sys/types.h> header shall define at least the following types:

13658	blkcnt_t	Used for file block counts.
13659	blksize_t	Used for block sizes.
13660	clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see
13661		<time.h>.
13662	clockid_t	Used for clock ID type in the clock and timer functions.
13663	dev_t	Used for device IDs.
13664	fsblkcnt_t	Used for file system block counts.
13665	fsfilcnt_t	Used for file system file counts.
13666	gid_t	Used for group IDs.
13667	id_t	Used as a general identifier; can be used to contain at least a pid_t ,
13668		uid_t , or gid_t .
13669	ino_t	Used for file serial numbers.
13670	XSI key_t	Used for XSI interprocess communication.
13671	mode_t	Used for some file attributes.
13672	nlink_t	Used for link counts.
13673	off_t	Used for file sizes.
13674	pid_t	Used for process IDs and process group IDs.
13675	pthread_attr_t	Used to identify a thread attribute object.
13676	pthread_barrier_t	Used to identify a barrier.
13677	pthread_barrierattr_t	Used to define a barrier attributes object.
13678	pthread_cond_t	Used for condition variables.
13679	pthread_condattr_t	Used to identify a condition attribute object.
13680	pthread_key_t	Used for thread-specific data keys.
13681	pthread_mutex_t	Used for mutexes.
13682	pthread_mutexattr_t	Used to identify a mutex attribute object.
13683	pthread_once_t	Used for dynamic package initialization.
13684	pthread_rwlock_t	Used for read-write locks.
13685	pthread_rwlockattr_t	Used for read-write lock attributes.
13686	pthread_spinlock_t	Used to identify a spin lock.

13687		pthread_t	Used to identify a thread.
13688		size_t	Used for sizes of objects.
13689		ssize_t	Used for a count of bytes or an error indication.
13690		suseconds_t	Used for time in microseconds.
13691		time_t	Used for time in seconds.
13692		timer_t	Used for timer ID returned by <i>timer_create()</i> .
13693	OB TRC	trace_attr_t	Used to identify a trace stream attributes object
13694	OB TRC	trace_event_id_t	Used to identify a trace event type.
13695	OB TEF	trace_event_set_t	Used to identify a trace event type set.
13696	OB TRC	trace_id_t	Used to identify a trace stream.
13697		uid_t	Used for user IDs.
13698		All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:	
13699			
13700		pthread_attr_t	
13701		pthread_barrier_t	
13702		pthread_barrierattr_t	
13703		pthread_cond_t	
13704		pthread_condattr_t	
13705		pthread_key_t	
13706		pthread_mutex_t	
13707		pthread_mutexattr_t	
13708		pthread_once_t	
13709		pthread_rwlock_t	
13710		pthread_rwlockattr_t	
13711		pthread_spinlock_t	
13712		pthread_t	
13713		timer_t	
13714	OB TRC	trace_attr_t	
13715		trace_event_id_t	
13716	OB TEF	trace_event_set_t	
13717	OB TRC	trace_id_t	

13718 Additionally:

- 13719 **mode_t** shall be an integer type.
- 13720 **dev_t** shall be an integer type.
- 13721 **nlink_t**, **uid_t**, **gid_t**, and **id_t** shall be integer types.
- 13722 **blkcnt_t** and **off_t** shall be signed integer types.
- 13723 **fsblkcnt_t**, **fsfilcnt_t**, and **ino_t** shall be defined as unsigned integer types.
- 13724 **size_t** shall be an unsigned integer type.
- 13725 **blksize_t**, **pid_t**, and **ssize_t** shall be signed integer types.

13726 CX **clock_t** shall be an integer or real-floating type. **time_t** shall be an integer type.

13727 The type **ssize_t** shall be capable of storing values at least in the range [-1, {SSIZE_MAX}].

13728 XSI The type **suseconds_t** shall be a signed integer type capable of storing values at least in the
13729 range [-1, 1 000 000].

13730 The implementation shall support one or more programming environments in which the widths
13731 of **blksize_t**, **pid_t**, **size_t**, **ssize_t**, and **suseconds_t** are no greater than the width of type **long**.
13732 The names of these programming environments can be obtained using the *confstr()* function or
13733 the *getconf* utility.

13734 There are no defined comparison or assignment operators for the following types:

13735 **pthread_attr_t**
13736 **pthread_barrier_t**
13737 **pthread_barrierattr_t**
13738 **pthread_cond_t**
13739 **pthread_condattr_t**
13740 **pthread_mutex_t**
13741 **pthread_mutexattr_t**
13742 **pthread_rwlock_t**
13743 **pthread_rwlockattr_t**
13744 **pthread_spinlock_t**
13745 **timer_t**
13746 OB TRC **trace_attr_t**

13747 **APPLICATION USAGE**

13748 None.

13749 **RATIONALE**

13750 None.

13751 **FUTURE DIRECTIONS**

13752 None.

13753 **SEE ALSO**

13754 [<time.h>](#)

13755 XSH *confstr()*

13756 XCU *getconf*

13757 **CHANGE HISTORY**

13758 First released in Issue 1. Derived from Issue 1 of the SVID.

13759 **Issue 5**

13760 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

13761 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

13762 Large File System extensions are added.

13763 Updated for alignment with the POSIX Threads Extension.

13764 **Issue 6**

13765 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
13766 alignment with IEEE Std 1003.1j-2000.

13767 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and

- 13768 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
13769 option. The threads types are marked THR.
- 13770 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread_t** to the list
13771 of types that are not required to be arithmetic types, thus allowing **pthread_t** to be defined as a
13772 structure.
- 13773 **Issue 7**
- 13774 Austin Group Interpretation 1003.1-2001 #033 is applied, requiring **key_t** to be an arithmetic
13775 type.
- 13776 The Trace option types are marked obsolescent.
- 13777 The **clock_t** and **id_t** types are moved from the XSI option to the Base.
- 13778 The **pthread_barrier_t** and **pthread_barrierattr_t** types are moved from the Barriers option to
13779 the Base.
- 13780 The **pthread_spinlock_t** type is moved from the Spin Locks option to the Base.
- 13781 Functionality relating to the Timers and Threads options is moved to the Base.
- 13782 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0069 [210], XBD/TC1-2008/0070 [28],
13783 XBD/TC1-2008/0071 [376], XBD/TC1-2008/0072 [210], and XBD/TC1-2008/0073 [327] are
13784 applied.
- 13785 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0079 [856] and XBD/TC2-2008/0080
13786 [659] are applied.

13787 **NAME**

13788 sys/uio.h ‡definitions for vector I/O operations

13789 **SYNOPSIS**

13790 XSI #include <sys/uio.h>

13791 **DESCRIPTION**13792 The **<sys/uio.h>** header shall define the **iovec** structure, which shall include at least the
13793 following members:

13794 void *iov_base Base address of a memory region for input or output.

13795 size_t iov_len The size of the memory pointed to by *iov_base*.13796 The **<sys/uio.h>** header uses the **iovec** structure for scatter/gather I/O.13797 The **<sys/uio.h>** header shall define the **ssize_t** and **size_t** types as described in **<sys/types.h>**.13798 The following shall be declared as functions and may also be defined as macros. Function
13799 prototypes shall be provided.

13800 ssize_t readv(int, const struct iovec *, int);

13801 ssize_t writev(int, const struct iovec *, int);

13802 **APPLICATION USAGE**13803 The implementation can put a limit on the number of scatter/gather elements which can be
13804 processed in one call. The symbol {IOV_MAX} defined in **<limits.h>** should always be used to
13805 learn about the limits instead of assuming a fixed value.13806 **RATIONALE**13807 Traditionally, the maximum number of scatter/gather elements the system can process in one
13808 call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-2001 this value is
13809 replaced by the constant {IOV_MAX} which can be found in **<limits.h>**.13810 **FUTURE DIRECTIONS**

13811 None.

13812 **SEE ALSO**13813 **<limits.h>**, **<sys/types.h>**13814 XSH *read()*, *readv()*, *write()*, *writev()*13815 **CHANGE HISTORY**

13816 First released in Issue 4, Version 2.

13817 **Issue 6**

13818 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13819 **NAME**

13820 sys/un.h ‡definitions for UNIX domain sockets

13821 **SYNOPSIS**

13822 #include <sys/un.h>

13823 **DESCRIPTION**13824 The <sys/un.h> header shall define the **sockaddr_un** structure, which shall include at least the
13825 following members:13826 sa_family_t sun_family Address family.
13827 char sun_path[] Socket pathname.13828 The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Pointers to this
13829 type shall be cast by applications to **struct sockaddr *** for use with socket functions.13830 The <sys/un.h> header shall define the **sa_family_t** type as described in <sys/socket.h>.13831 **APPLICATION USAGE**13832 The size of *sun_path* has intentionally been left undefined. This is because different
13833 implementations use different sizes. For example, 4.3 BSD uses a size of 108, and 4.4 BSD uses a
13834 size of 104. Since most implementations originate from BSD versions, the size is typically in the
13835 range 92 to 108.13836 Applications should not assume a particular length for *sun_path* or assume that it can hold
13837 {_POSIX_PATH_MAX} bytes (256).13838 Although applications are required to initialize all members (including any non-standard ones)
13839 of a **sockaddr_in6** structure (see <netinet/in.h>, on page 306), the same is not required for the
13840 **sockaddr_un** structure, since historically many applications only initialized the standard
13841 members. Despite this, applications are encouraged to initialize **sockaddr_un** structures in a
13842 manner similar to the required initialization of **sockaddr_in6** structures.13843 **RATIONALE**

13844 None.

13845 **FUTURE DIRECTIONS**

13846 None.

13847 **SEE ALSO**

13848 <netinet/in.h>, <sys/socket.h>

13849 XSH *bind()*, *socket()*, *socketpair()*13850 **CHANGE HISTORY**

13851 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13852 **Issue 7**

13853 The value for {_POSIX_PATH_MAX} is updated to 256.

13854 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0074 [355] is applied.

13855 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0081 [934] is applied.

13856 **NAME**

13857 sys/utsname.h — system name structure

13858 **SYNOPSIS**

13859 #include <sys/utsname.h>

13860 **DESCRIPTION**13861 The **<sys/utsname.h>** header shall define the structure **utsname** which shall include at least the
13862 following members:

13863 char sysname[] Name of this implementation of the operating system.
13864 char nodename[] Name of this node within the communications
13865 network to which this node is attached, if any.
13866 char release[] Current release level of this implementation.
13867 char version[] Current version level of this release.
13868 char machine[] Name of the hardware type on which the system is running.

13869 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
13870 null byte.

13871 The following shall be declared as a function and may also be defined as a macro:

13872 int uname(struct utsname *);

13873 **APPLICATION USAGE**

13874 None.

13875 **RATIONALE**

13876 None.

13877 **FUTURE DIRECTIONS**

13878 None.

13879 **SEE ALSO**13880 XSH *uname()*13881 **CHANGE HISTORY**

13882 First released in Issue 1. Derived from Issue 1 of the SVID.

13883 **Issue 6**

13884 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of
13885 *nodename* within the **utsname** structure from “an implementation-defined communications
13886 network” to “the communications network to which this node is attached, if any”.

13887 **NAME**

13888 sys/wait.h ‡declarations for waiting

13889 **SYNOPSIS**

13890 #include <sys/wait.h>

13891 **DESCRIPTION**

13892 The <sys/wait.h> header shall define the following symbolic constants for use with *waitpid()*:

- 13893 XSI **WCONTINUED** Report status of continued child process.
- 13894 **WNOHANG** Do not hang if no status is available; return immediately.
- 13895 **WUNTRACED** Report status of stopped child process.

13896 The <sys/wait.h> header shall define the following macros for analysis of process status values:

- 13897 **WEXITSTATUS** Return exit status.
- 13898 XSI **WIFCONTINUED** True if child has been continued.
- 13899 **WIFEXITED** True if child exited normally.
- 13900 **WIFSIGNALED** True if child exited due to uncaught signal.
- 13901 **WIFSTOPPED** True if child is currently stopped.
- 13902 **WSTOPSIG** Return signal number that caused process to stop.
- 13903 **WTERMSIG** Return signal number that caused process to terminate.

13904 The <sys/wait.h> header shall define the following symbolic constants as possible values for the *options* argument to *waitid()*:

- 13906 **WEXITED** Wait for processes that have exited.
- 13907 **WNOWAIT** Keep the process whose status is returned in *infop* in a waitable state.
- 13908 **WSTOPPED** Status is returned for any child that has stopped upon receipt of a signal.

13909 XSI The **WCONTINUED** and **WNOHANG** constants, described above for *waitpid()*, can also be used with *waitid()*.

13911 The type **idtype_t** shall be defined as an enumeration type whose possible values shall include at least the following:

- 13913 **P_ALL**
- 13914 **P_PGID**
- 13915 **P_PID**

13916 The <sys/wait.h> header shall define the **id_t** and **pid_t** types as described in <sys/types.h>.

13917 The <sys/wait.h> header shall define the **siginfo_t** type and the **sigval** union as described in <signal.h>.

13919 Inclusion of the <sys/wait.h> header may also make visible all symbols from <signal.h>.

13920 The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
13922 pid_t wait(int *);
13923 int waitid(idtype_t, id_t, siginfo_t *, int);
13924 pid_t waitpid(pid_t, int *, int);
```


13925 **APPLICATION USAGE**

13926 None.

13927 **RATIONALE**

13928 None.

13929 **FUTURE DIRECTIONS**

13930 None.

13931 **SEE ALSO**13932 [<signal.h>](#), [<sys/resource.h>](#), [<sys/types.h>](#)13933 XSH *wait()*, *waitid()*13934 **CHANGE HISTORY**

13935 First released in Issue 3.

13936 Included for alignment with the POSIX.1-1988 standard.

13937 **Issue 6**13938 The *wait3()* function is removed.13939 **Issue 7**13940 The *waitid()* function and symbolic constants for its *options* argument are moved to the Base.

13941 The description of the WNOHANG constant is clarified.

13942 The requirement for **<sys/wait.h>** to define the **rusage** structure as described in
13943 **<sys/resource.h>** is removed, and **<sys/wait.h>** is no longer allowed to make visible all symbols
13944 from **<sys/resource.h>**.13945 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0082 [579] and XBD/TC2-2008/0083
13946 [564] are applied.

13947 **NAME**

13948 syslog.h — definitions for system error logging

13949 **SYNOPSIS**

13950 XSI #include <syslog.h>

13951 **DESCRIPTION**

13952 The <syslog.h> header shall define the following symbolic constants, zero or more of which
13953 may be OR'ed together to form the *logopt* option of *openlog()*:

13954 LOG_PID Log the process ID with each message.

13955 LOG_CONS Log to the system console on error.

13956 LOG_NDELAY Connect to syslog daemon immediately.

13957 LOG_ODELAY Delay open until *syslog()* is called.

13958 LOG_NOWAIT Do not wait for child processes.

13959 The <syslog.h> header shall define the following symbolic constants for use as the *facility*
13960 argument to *openlog()*:

13961 LOG_KERN Reserved for message generated by the system.

13962 LOG_USER Message generated by a process.

13963 LOG_MAIL Reserved for message generated by mail system.

13964 LOG_NEWS Reserved for message generated by news system.

13965 LOG_UUCP Reserved for message generated by UUCP system.

13966 LOG_DAEMON Reserved for message generated by system daemon.

13967 LOG_AUTH Reserved for message generated by authorization daemon.

13968 LOG_CRON Reserved for message generated by clock daemon.

13969 LOG_LPR Reserved for message generated by printer system.

13970 LOG_LOCAL0 Reserved for local use.

13971 LOG_LOCAL1 Reserved for local use.

13972 LOG_LOCAL2 Reserved for local use.

13973 LOG_LOCAL3 Reserved for local use.

13974 LOG_LOCAL4 Reserved for local use.

13975 LOG_LOCAL5 Reserved for local use.

13976 LOG_LOCAL6 Reserved for local use.

13977 LOG_LOCAL7 Reserved for local use.

13978 The <syslog.h> header shall define the following macros for constructing the *maskpri* argument
13979 to *setlogmask()*. The following macros expand to an expression of type **int** when the argument
13980 *pri* is an expression of type **int**:

13981 LOG_MASK(*pri*) A mask for priority *pri*.

13982 The <syslog.h> header shall define the following symbolic constants for use as the *priority*
13983 argument of *syslog()*:

13984	LOG_EMERG	A panic condition was reported to all processes.
13985	LOG_ALERT	A condition that should be corrected immediately.
13986	LOG_CRIT	A critical condition.
13987	LOG_ERR	An error message.
13988	LOG_WARNING	A warning message.
13989	LOG_NOTICE	A condition requiring special handling.
13990	LOG_INFO	A general information message.
13991	LOG_DEBUG	A message useful for debugging programs.
13992	The following shall be declared as functions and may also be defined as macros. Function	
13993	prototypes shall be provided.	
13994	<code>void closelog(void);</code>	
13995	<code>void openlog(const char *, int, int);</code>	
13996	<code>int setlogmask(int);</code>	
13997	<code>void syslog(int, const char *, ...);</code>	
13998	APPLICATION USAGE	
13999	None.	
14000	RATIONALE	
14001	None.	
14002	FUTURE DIRECTIONS	
14003	None.	
14004	SEE ALSO	
14005	XSH closelog()	
14006	CHANGE HISTORY	
14007	First released in Issue 4, Version 2.	
14008	Issue 5	
14009	Moved from X/Open UNIX to BASE.	
14010	Issue 7	
14011	This reference page is clarified with respect to macros and symbolic constants.	

14012 **NAME**
 14013 tar.h — extended tar definitions

14014 **SYNOPSIS**
 14015 #include <tar.h>

14016 **DESCRIPTION**
 14017 The <tar.h> header shall define the following symbolic constants with the indicated values.
 14018 General definitions:

Name	Value	Description
TMAGIC	"ustar"	Used in the magic field in the ustar header block, including the trailing null byte.
TMAGLEN	6	Length in octets of the magic field.
TVERSION	"00"	Used in the version field in the ustar header block, excluding the trailing null byte.
TVERSLEN	2	Length in octets of the version field.

14026 *Typeflag* field definitions:

Name	Value	Description
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

14037 *Mode* field bit definitions (octal):

Name	Value	Description
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

14051 APPLICATION USAGE

14052 None.

14053 RATIONALE

14054 None.

14055 FUTURE DIRECTIONS

14056 None.

14057 SEE ALSO

14058 XCU *pax*

14059 CHANGE HISTORY

14060 First released in Issue 3. Derived from the POSIX.1-1988 standard.

14061 Issue 6

14062 The SEE ALSO section is updated to refer to *pax*.

14063 Issue 7

14064 This reference page is clarified with respect to macros and symbolic constants.

14065 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0084 [707] is applied.

14066 **NAME**

14067 `termios.h` ‡define values for termios

14068 **SYNOPSIS**

14069 `#include <termios.h>`

14070 **DESCRIPTION**

14071 The <termios.h> header shall contain the definitions used by the terminal I/O interfaces (see
14072 [Chapter 11](#) (on page 199) for the structures and names defined).

14073 **The termios Structure**

14074 The <termios.h> header shall define the following data types through **typedef**:

14075 **cc_t** Used for terminal special characters.

14076 **speed_t** Used for terminal baud rates.

14077 **tcflag_t** Used for terminal modes.

14078 The above types shall be all unsigned integer types.

14079 The implementation shall support one or more programming environments in which the widths
14080 of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these
14081 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

14082 The <termios.h> header shall define the **termios** structure, which shall include at least the
14083 following members:

- 14084 `tcflag_t c_iflag` Input modes.
- 14085 `tcflag_t c_oflag` Output modes.
- 14086 `tcflag_t c_cflag` Control modes.
- 14087 `tcflag_t c_lflag` Local modes.
- 14088 `cc_t c_cc[NCCS]` Control characters.

14089 The <termios.h> header shall define the following symbolic constant:

14090 **NCCS** Size of the array `c_cc` for control characters.

14091 The <termios.h> header shall define the following symbolic constants for use as subscripts for
14092 the array `c_cc`:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

14106 The subscript values shall be suitable for use in **#if** preprocessing directives and shall be distinct,
14107 except that the VMIN and VTIME subscripts may have the same values as the VEOF and VEOL
14108 subscripts, respectively.

14109 **Input Modes**

14110 The **<termios.h>** header shall define the following symbolic constants for use as flags in the
 14111 *c_iflag* field. The *c_iflag* field describes the basic terminal input control.

14112	BRKINT	Signal interrupt on break.
14113	ICRNL	Map CR to NL on input.
14114	IGNBRK	Ignore break condition.
14115	IGNCR	Ignore CR.
14116	IGNPAR	Ignore characters with parity errors.
14117	INLCR	Map NL to CR on input.
14118	INPCK	Enable input parity check.
14119	ISTRIP	Strip character.
14120	IXANY	Enable any character to restart output.
14121	IXOFF	Enable start/stop input control.
14122	IXON	Enable start/stop output control.
14123	PARMRK	Mark parity errors.

14124 **Output Modes**

14125 The **<termios.h>** header shall define the following symbolic constants for use as flags in the
 14126 *c_oflag* field. The *c_oflag* field specifies the system treatment of output.

14127	OPOST	Post-process output.
14128	XSI	ONLCR Map NL to CR-NL on output.
14129	XSI	OCRNL Map CR to NL on output.
14130	XSI	ONOCR No CR output at column 0.
14131	XSI	ONLRET NL performs CR function.
14132	XSI	OFDEL Fill is DEL.
14133	XSI	OFILL Use fill characters for delay.
14134	XSI	NLDLY Select newline delays:
14135		NL0 Newline type 0.
14136		NL1 Newline type 1.
14137	XSI	CRDLY Select carriage-return delays:
14138		CR0 Carriage-return delay type 0.
14139		CR1 Carriage-return delay type 1.
14140		CR2 Carriage-return delay type 2.
14141		CR3 Carriage-return delay type 3.

14142	XSI	TABDLY	Select horizontal-tab delays:
14143		TAB0	Horizontal-tab delay type 0.
14144		TAB1	Horizontal-tab delay type 1.
14145		TAB2	Horizontal-tab delay type 2.
14146		TAB3	Expand tabs to spaces.
14147	XSI	BSDLY	Select backspace delays:
14148		BS0	Backspace-delay type 0.
14149		BS1	Backspace-delay type 1.
14150	XSI	VTDLY	Select vertical-tab delays:
14151		VT0	Vertical-tab delay type 0.
14152		VT1	Vertical-tab delay type 1.
14153	XSI	FFDLY	Select form-feed delays:
14154		FF0	Form-feed delay type 0.
14155		FF1	Form-feed delay type 1.

14156 **Baud Rate Selection**

14157 The <termios.h> header shall define the following symbolic constants for use as values of
 14158 objects of type **speed_t**.

14159 The input and output baud rates are stored in the **termios** structure. These are the valid values
 14160 for objects of type **speed_t**. Not all baud rates need be supported by the underlying hardware.

14161	B0	Hang up
14162	B50	50 baud
14163	B75	75 baud
14164	B110	110 baud
14165	B134	134.5 baud
14166	B150	150 baud
14167	B200	200 baud
14168	B300	300 baud
14169	B600	600 baud
14170	B1200	1 200 baud
14171	B1800	1 800 baud
14172	B2400	2 400 baud

14173	B4800	4 800 baud
14174	B9600	9 600 baud
14175	B19200	19 200 baud
14176	B38400	38 400 baud

14177 Control Modes

14178 The **<termios.h>** header shall define the following symbolic constants for use as flags in the
14179 *c_cflag* field. The *c_cflag* field describes the hardware control of the terminal; not all values
14180 specified are required to be supported by the underlying hardware.

14181	CSIZE	Character size:
14182		CS5 5 bits
14183		CS6 6 bits
14184		CS7 7 bits
14185		CS8 8 bits
14186	CSTOPB	Send two stop bits, else one.
14187	CREAD	Enable receiver.
14188	PARENB	Parity enable.
14189	PARODD	Odd parity, else even.
14190	HUPCL	Hang up on last close.
14191	CLOCAL	Ignore modem status lines.

14192 The implementation shall support the functionality associated with the symbols CS7, CS8,
14193 CSTOPB, PARODD, and PARENB.

14194 Local Modes

14195 The **<termios.h>** header shall define the following symbolic constants for use as flags in the
14196 *c_lflag* field. The *c_lflag* field of the argument structure is used to control various terminal
14197 functions.

14198	ECHO	Enable echo.
14199	ECHOE	Echo erase character as error-correcting backspace.
14200	ECHOK	Echo KILL.
14201	ECHONL	Echo NL.
14202	ICANON	Canonical input (erase and kill processing).
14203	IEXTEN	Enable extended input character processing.
14204	ISIG	Enable signals.
14205	NOFLSH	Disable flush after interrupt or quit.
14206	TOSTOP	Send SIGTTOU for background output.

14207 **Attribute Selection**

14208 The <termios.h> header shall define the following symbolic constants for use with *tcsetattr()*:

- 14209 TCSANOW Change attributes immediately.
- 14210 TCSADRAIN Change attributes when output has drained.
- 14211 TCSAFLUSH Change attributes when output has drained; also flush pending input.

14212 **Line Control**

14213 The <termios.h> header shall define the following symbolic constants for use with *tcflush()*:

- 14214 TCIFLUSH Flush pending input.
- 14215 TCIOFLUSH Flush both pending input and untransmitted output.
- 14216 TCOFLUSH Flush untransmitted output.

14217 The <termios.h> header shall define the following symbolic constants for use with *tcflow()*:

- 14218 TCIOFF Transmit a STOP character, intended to suspend input data.
- 14219 TCION Transmit a START character, intended to restart input data.
- 14220 TCOOFF Suspend output.
- 14221 TCOON Restart output.

14222 The <termios.h> header shall define the **pid_t** type as described in <sys/types.h>.

14223 The following shall be declared as functions and may also be defined as macros. Function
14224 prototypes shall be provided.

```

14225 speed_t cfgetispeed(const struct termios *);
14226 speed_t cfgetospeed(const struct termios *);
14227 int cfsetispeed(struct termios *, speed_t);
14228 int cfsetospeed(struct termios *, speed_t);
14229 int tcdrain(int);
14230 int tcflow(int, int);
14231 int tcflush(int, int);
14232 int tcgetattr(int, struct termios *);
14233 pid_t tcgetsid(int);
14234 int tcsendbreak(int, int);
14235 int tcsetattr(int, int, const struct termios *);
    
```

14236 **APPLICATION USAGE**

14237 The following names are reserved for XSI-conformant systems to use as an extension to the
14238 above; therefore strictly conforming applications shall not use them:

- 14239 CBAUD EXTB VDSUSP
- 14240 DEFCHO FLUSHO VLNEXT
- 14241 ECHOCTL LOBLK VREPRINT
- 14242 ECHOK E PENDIN VSTATUS
- 14243 ECHOPRT SWTCH VWERASE
- 14244 EXTA VDISCARD

- 14245 **RATIONALE**
14246 None.
- 14247 **FUTURE DIRECTIONS**
14248 None.
- 14249 **SEE ALSO**
14250 [<sys/types.h>](#)
14251 XSH [cfgetispeed\(\)](#), [cfgetospeed\(\)](#), [cfsetispeed\(\)](#), [cfsetospeed\(\)](#), [confstr\(\)](#), [tcdrain\(\)](#), [tcflow\(\)](#), [tcflush\(\)](#),
14252 [tcgetattr\(\)](#), [tcgetsid\(\)](#), [tcsendbreak\(\)](#), [tcsetattr\(\)](#)
14253 XCU [Chapter 11](#) (on page 199), [getconf](#)
- 14254 **CHANGE HISTORY**
14255 First released in Issue 3.
14256 Included for alignment with the ISO POSIX-1 standard.
- 14257 **Issue 6**
14258 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.
14259 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are
14260 reaffirmed.
14261 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to
14262 ECHOKE in the APPLICATION USAGE section.
- 14263 **Issue 7**
14264 Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the
14265 IXANY symbol from the XSI option to the Base.
14266 SD5-XBD-ERN-35 is applied, adding the OFDEL output mode.
14267 This reference page is clarified with respect to macros and symbolic constants, and a declaration
14268 for the `pid_t` type is added.

14269 **NAME**

14270 tgmath.h — type-generic macros

14271 **SYNOPSIS**

14272 #include <tgmath.h>

14273 **DESCRIPTION**

14274 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 14275 conflict between the requirements described here and the ISO C standard is unintentional. This
 14276 volume of POSIX.1-2017 defers to the ISO C standard.

14277 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define
 14278 several type-generic macros.

14279 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**)
 14280 or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
 14281 XSI **double**. For each such function, except *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*,
 14282 there shall be a corresponding type-generic macro. The parameters whose corresponding real type is
 14283 **double** in the function synopsis are generic parameters. Use of the macro invokes a function
 14284 whose corresponding real type and type domain are determined by the arguments for the
 14285 generic parameters.

14286 Use of the macro invokes a function whose generic parameters have the corresponding real type
 14287 determined as follows:

14288 First, if any argument for generic parameters has type **long double**, the type determined is
 14289 **long double**.

14290 Otherwise, if any argument for generic parameters has type **double** or is of integer type,
 14291 the type determined is **double**.

14292 Otherwise, the type determined is **float**.

14293 For each unsuffixed function in the <math.h> header for which there is a function in the
 14294 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic
 14295 macro (for both functions) has the same name as the function in the <math.h> header. The
 14296 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

	<math.h> Function	<complex.h> Function	Type-Generic Macro
14297			
14298	<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
14299	<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
14300	<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
14301	<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
14302	<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
14303	<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
14304	<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
14305	<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
14306	<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
14307	<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
14308	<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
14309	<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
14310	<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
14311	<i>log()</i>	<i>clog()</i>	<i>log()</i>
14312	<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
14313	<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
14314	<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>

14315 If at least one argument for a generic parameter is complex, then use of the macro invokes a
14316 complex function; otherwise, use of the macro invokes a real function.

14317 For each unsuffixed function in the <math.h> header without a c-prefixed counterpart in the
14318 XSI <complex.h> header, except for *modf()*, *j0()*, *j1()*, *jn()*, *y0()*, *y1()*, and *yn()*, the corresponding
14319 type-generic macro has the same name as the function. These type-generic macros are:

14320	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
14321	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
14322	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
14323	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
14324	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbln()</i>
14325	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbn()</i>
14326	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
14327	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
14328	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
14329	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

14330 If all arguments for generic parameters are real, then use of the macro invokes a real function;
14331 otherwise, use of the macro results in undefined behavior.

14332 For each unsuffixed function in the <complex.h> header that is not a c-prefixed counterpart to a
14333 function in the <math.h> header, the corresponding type-generic macro has the same name as
14334 the function. These type-generic macros are:

14335	<i>carg()</i>
14336	<i>cimag()</i>
14337	<i>conj()</i>
14338	<i>cproj()</i>
14339	<i>creal()</i>

14340 Use of the macro with any real or complex argument invokes a complex function.

14341 **APPLICATION USAGE**

14342 With the declarations:

```

14343 #include <tgmath.h>
14344 int n;
14345 float f;
14346 double d;
14347 long double ld;
14348 float complex fc;
14349 double complex dc;
14350 long double complex ldc;

```

14351 functions invoked by use of type-generic macros are shown in the following table:

Macro	Use Invokes
<i>exp(n)</i>	<i>exp(n)</i> , the function
<i>acosh(f)</i>	<i>acoshf(f)</i>
<i>sin(d)</i>	<i>sin(d)</i> , the function
<i>atan(ld)</i>	<i>atanl(ld)</i>
<i>log(fc)</i>	<i>clogf(fc)</i>
<i>sqrt(dc)</i>	<i>csqrt(dc)</i>

14359
14360
14361
14362
14363
14364
14365
14366
14367
14368
14369
14370
14371
14372
14373
14374

Macro	Use Invokes
<i>pow(ldc,f)</i>	<i>cpowl(ldc, f)</i>
<i>remainder(n,n)</i>	<i>remainder(n, n)</i> , the function
<i>nextafter(d,f)</i>	<i>nextafter(d, f)</i> , the function
<i>nexttoward(f,ld)</i>	<i>nexttowardf(f, ld)</i>
<i>copysign(n,ld)</i>	<i>copysignl(n, ld)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(dc)</i>	Undefined behavior
<i>fmax(ldc,ld)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(ld)</i>	<i>cimagl(ld)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(dc)</i>	<i>carg(dc)</i> , the function
<i>cproj(ldc)</i>	<i>cprojl(ldc)</i>

RATIONALE

14375
14376
14377
14378
14379
14380
14381
14382
14383
14384
14385
14386
14387
14388
14389
14390
14391
14392
14393
14394
14395
14396
14397
14398
14399
14400
14401
14402
14403
14404

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic *cos()* macro, the expression *cos((float)x)* will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

The macros are placed in their own header in order not to silently break old programs that include the <math.h> header; for example, with:

```
printf ("%e", sin(x))
```

*modf(double, double *)* is excluded because no way was seen to make it safe without complicating the type resolution.

The implementation might, as an extension, endow appropriate ones of the macros that POSIX.1-2017 specifies only for real arguments with the ability to invoke the complex functions.

POSIX.1-2017 does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for example, could be implemented with:

```
#undef sqrt
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

The great majority of existing C programs are expected to be unaffected when the <tgmath.h> header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return

14405 values differ.

14406 The ability to overload on integer as well as floating types would have been useful for some
14407 functions; for example, *copysign()*. Overloading with different numbers of arguments would
14408 have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities
14409 would have complicated the specification; and their natural consistent use, such as for a floating
14410 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the
14411 ISO/IEC 9899:1999 standard for insufficient benefit.

14412 The ISO C standard in no way limits the implementation's options for efficiency, including
14413 inlining library functions.

14414 **FUTURE DIRECTIONS**

14415 None.

14416 **SEE ALSO**

14417 [<math.h>](#), [<complex.h>](#)

14418 XSH *cabs()*, *fabs()*, *modf()*

14419 **CHANGE HISTORY**

14420 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

14421 **Issue 7**

14422 Austin Group Interpretation 1003.1-2001 #184 is applied, clarifying the functions for which a
14423 corresponding type-generic macro exists with the same name as the function.

14424 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0075 [357,427] is applied.

14425 **NAME**

14426 time.h ‡time types

14427 **SYNOPSIS**

14428 #include <time.h>

14429 **DESCRIPTION**

14430 CX Some of the functionality described on this reference page extends the ISO C standard.
 14431 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 14432 enable the visibility of these symbols in this header.

14433 The <time.h> header shall define the **clock_t**, **size_t**, **time_t**, types as described in
 14434 <sys/types.h>.

14435 CX The <time.h> header shall define the **clockid_t** and **timer_t** types as described in <sys/types.h>.

14436 The <time.h> header shall define the **locale_t** type as described in <locale.h>.

14437 CPT The <time.h> header shall define the **pid_t** type as described in <sys/types.h>.

14438 CX The tag **sigevent** shall be declared as naming an incomplete structure type, the contents of which
 14439 are described in the <signal.h> header.

14440 The <time.h> header shall declare the **tm** structure, which shall include at least the following
 14441 members:

```

14442 int      tm_sec   Seconds [0,60].
14443 int      tm_min   Minutes [0,59].
14444 int      tm_hour   Hour [0,23].
14445 int      tm_mday   Day of month [1,31].
14446 int      tm_mon    Month of year [0,11].
14447 int      tm_year   Years since 1900.
14448 int      tm_wday   Day of week [0,6] (Sunday =0).
14449 int      tm_yday   Day of year [0,365].
14450 int      tm_isdst  Daylight Savings flag.
```

14451 The value of *tm_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings
 14452 Time is not in effect, and negative if the information is not available.

14453 CX The <time.h> header shall declare the **timespec** structure, which shall include at least the
 14454 following members:

```

14455 time_t   tv_sec    Seconds.
14456 long     tv_nsec   Nanoseconds.
```

14457 The <time.h> header shall also declare the **itimerspec** structure, which shall include at least the
 14458 following members:

```

14459 struct timespec it_interval  Timer period.
14460 struct timespec it_value    Timer expiration.
```

14461 The <time.h> header shall define the following macros:

14462 **NULL** As described in <stddef.h>.

14463 **CLOCKS_PER_SEC** A number used to convert the value returned by the *clock()* function into
 14464 XSI seconds. The value shall be an expression with type **clock_t**. The value of
 14465 **CLOCKS_PER_SEC** shall be 1 million on XSI-conformant systems.
 14466 However, it may be variable on other systems, and it should not be

14467 assumed that `CLOCKS_PER_SEC` is a compile-time constant.

14468 CX The **<time.h>** header shall define the following symbolic constants. The values shall have a type
14469 that is assignment-compatible with `clockid_t`.

14470 MON **CLOCK_MONOTONIC**
14471 The identifier for the system-wide monotonic clock, which is defined as a
14472 clock measuring real time, whose value cannot be set via `clock_settime()`
14473 and which cannot have negative clock jumps. The maximum possible
14474 clock jump shall be implementation-defined.

14475 CPT **CLOCK_PROCESS_CPUTIME_ID**
14476 The identifier of the CPU-time clock associated with the process making a
14477 `clock()` or `timer*()` function call.

14478 CX **CLOCK_REALTIME** The identifier of the system-wide clock measuring real time.

14479 TCT **CLOCK_THREAD_CPUTIME_ID**
14480 The identifier of the CPU-time clock associated with the thread making a
14481 `clock()` or `timer*()` function call.

14482 CX The **<time.h>** header shall define the following symbolic constant:
14483 **TIMER_ABSTIME** Flag indicating time is absolute. For functions taking timer objects, this
14484 refers to the clock associated with the timer.

14485 XSI The **<time.h>** header shall provide a declaration or definition for `getdate_err`. The `getdate_err`
14486 symbol shall expand to an expression of type `int`. It is unspecified whether `getdate_err` is a macro
14487 or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a
14488 macro definition is suppressed in order to access an actual object, or a program defines an
14489 identifier with the name `getdate_err`, the behavior is undefined.

14490 The following shall be declared as functions and may also be defined as macros. Function
14491 prototypes shall be provided.

14492 OB `char *asctime(const struct tm *);`
14493 OB CX `char *asctime_r(const struct tm *restrict, char *restrict);`
14494 `clock_t clock(void);`
14495 CPT `int clock_getcpu(clockid_t, clockid_t *);`
14496 CX `int clock_getres(clockid_t, struct timespec *);`
14497 `int clock_gettime(clockid_t, struct timespec *);`
14498 `int clock_nanosleep(clockid_t, int, const struct timespec *,`
14499 `struct timespec *);`
14500 `int clock_settime(clockid_t, const struct timespec *);`
14501 OB `char *ctime(const time_t *);`
14502 OB CX `char *ctime_r(const time_t *, char *);`
14503 `double difftime(time_t, time_t);`
14504 XSI `struct tm *getdate(const char *);`
14505 `struct tm *gmtime(const time_t *);`
14506 CX `struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);`
14507 `struct tm *localtime(const time_t *);`
14508 CX `struct tm *localtime_r(const time_t *restrict, struct tm *restrict);`
14509 `time_t mktime(struct tm *);`
14510 CX `int nanosleep(const struct timespec *, struct timespec *);`
14511 `size_t strftime(char *restrict, size_t, const char *restrict,`
14512 `const struct tm *restrict);`

```

14513 CX      size_t      strftime_l(char *restrict, size_t, const char *restrict,
14514          const struct tm *restrict, locale_t);
14515 XSI      char      *strptime(const char *restrict, const char *restrict,
14516          struct tm *restrict);
14517          time_t      time(time_t *);
14518 CX      int      timer_create(clockid_t, struct sigevent *restrict,
14519          timer_t *restrict);
14520          int      timer_delete(timer_t);
14521          int      timer_getoverrun(timer_t);
14522          int      timer_gettime(timer_t, struct itimerspec *);
14523          int      timer_settime(timer_t, int, const struct itimerspec *restrict,
14524          struct itimerspec *restrict);
14525          void      tzset(void);
    
```

14526 The <time.h> header shall declare the following as variables:

```

14527 XSI      extern int      daylight;
14528          extern long     timezone;
14529 CX      extern char     *tzname[];
    
```

14530 CX **Inclusion of the <time.h> header may make visible all symbols from the <signal.h> header.**

14531 **APPLICATION USAGE**

14532 The range [0,60] for *tm_sec* allows for the occasional leap second.
 14533 *tm_year* is a signed value; therefore, years before 1900 may be represented.
 14534 To obtain the number of clock ticks per second returned by the *times()* function, applications
 14535 should call *sysconf(_SC_CLK_TCK)*.

14536 **RATIONALE**

14537 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of
 14538 UTC does not permit double leap seconds, so all mention of double leap seconds has been
 14539 removed, and the range shortened from the former [0,61] seconds seen in earlier versions of this
 14540 standard.

14541 **FUTURE DIRECTIONS**

14542 None.

14543 **SEE ALSO**

14544 [<locale.h>](#), [<signal.h>](#), [<stddef.h>](#), [<sys/types.h>](#)
 14545 XSH Section 2.2 (on page 472), *asctime()*, *clock()*, *clock_getcpuclockid()*, *clock_getres()*,
 14546 *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *mq_receive()*,
 14547 *mq_send()*, *nanosleep()*, *pthread_getcpuclockid()*, *pthread_mutex_timedlock()*,
 14548 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *sem_timedwait()*, *strftime()*, *strptime()*,
 14549 *sysconf()*, *time()*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, *tzset()*, *utime()*

14550 **CHANGE HISTORY**

14551 First released in Issue 1. Derived from Issue 1 of the SVID.

14552 **Issue 5**

14553 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 14554 Threads Extension.

14555 Issue 6

14556 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t**
14557 and **timer_t** have been described.

14558 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

14559 The POSIX timer-related functions are marked as part of the Timers option.

14560 The symbolic name CLK_TCK is removed. Application usage is added describing how its
14561 equivalent functionality can be obtained using *sysconf()*.

14562 The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
14563 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

14564 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for
14565 alignment with IEEE Std 1003.1j-2000.

14566 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

14567 The range for seconds is changed from [0,61] to [0,60].

14568 The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*,
14569 *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

14570 IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the
14571 **<signal.h>** header may be made visible when the **<time.h>** header is included.

14572 Extensions beyond the ISO C standard are marked.

14573 Issue 7

14574 Austin Group Interpretation 1003.1-2001 #111 is applied.

14575 SD5-XBD-ERN-74 is applied.

14576 The *strftime_l()* function is added from The Open Group Technical Standard, 2006, Extended API
14577 Set Part 4.

14578 Functionality relating to the Timers option is moved to the Base.

14579 This reference page is clarified with respect to macros and symbolic constants, and declarations
14580 for the **locale_t** and **pid_t** types and the **sigevent** structure are added.

14581 The description of the *getdate_err* value is expanded.

14582 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0076 [212] and XBD/TC1-2008/0077
14583 [212] are applied.

14584 **NAME**

14585 trace.h ‡tracing

14586 **SYNOPSIS**

14587 OB TRC #include <trace.h>

14588 **DESCRIPTION**

14589 The <trace.h> header shall define the **posix_trace_event_info** structure, which shall include at
14590 least the following members:

```
14591           trace_event_id_t   posix_event_id
14592           pid_t                posix_pid
14593           void                 *posix_prog_address
14594           pthread_t            posix_thread_id
14595           struct timespec      posix_timestamp
14596           int                 posix_truncation_status
```

14597 The <trace.h> header shall define the **posix_trace_status_info** structure, which shall include at
14598 least the following members:

```
14599           int                 posix_stream_full_status
14600           int                 posix_stream_overrun_status
14601           int                 posix_stream_status
14602 OB TRL   int                 posix_log_full_status
14603           int                 posix_log_overrun_status
14604           int                 posix_stream_flush_error
14605           int                 posix_stream_flush_status
```

14606 The <trace.h> header shall define the following symbolic constants:

```
14607           POSIX_TRACE_ALL_EVENTS
14608 OB TRL   POSIX_TRACE_APPEND
14609 OB TRI   POSIX_TRACE_CLOSE_FOR_CHILD
14610 OB TEF   POSIX_TRACE_FILTER
14611 OB TRL   POSIX_TRACE_FLUSH
14612           POSIX_TRACE_FLUSH_START
14613           POSIX_TRACE_FLUSH_STOP
14614           POSIX_TRACE_FLUSHING
14615           POSIX_TRACE_FULL
14616           POSIX_TRACE_LOOP
14617           POSIX_TRACE_NO_OVERRUN
14618 OB TRL   POSIX_TRACE_NOT_FLUSHING
14619           POSIX_TRACE_NOT_FULL
14620 OB TRI   POSIX_TRACE_INHERITED
14621           POSIX_TRACE_NOT_TRUNCATED
14622           POSIX_TRACE_OVERFLOW
14623           POSIX_TRACE_OVERRUN
14624           POSIX_TRACE_RESUME
14625           POSIX_TRACE_RUNNING
14626           POSIX_TRACE_START
14627           POSIX_TRACE_STOP
14628           POSIX_TRACE_SUSPENDED
14629           POSIX_TRACE_SYSTEM_EVENTS
14630           POSIX_TRACE_TRUNCATED_READ
```

14631 POSIX_TRACE_TRUNCATED_RECORD
 14632 POSIX_TRACE_UNNAMED_USER_EVENT
 14633 POSIX_TRACE_UNTIL_FULL
 14634 POSIX_TRACE_WOPID_EVENTS

14635 OB TEF The **<trace.h>** header shall define the **size_t**, **trace_attr_t**, **trace_event_id_t**, **trace_event_set_t**,
 14636 and **trace_id_t** types as described in **<sys/types.h>**.

14637 The following shall be declared as functions and may also be defined as macros. Function
 14638 prototypes shall be provided.

```

14639 int posix_trace_attr_destroy(trace_attr_t *);
14640 int posix_trace_attr_getclockres(const trace_attr_t *,
14641     struct timespec *);
14642 int posix_trace_attr_getcreatetime(const trace_attr_t *,
14643     struct timespec *);
14644 int posix_trace_attr_getgenversion(const trace_attr_t *, char *);
14645 TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict,
14646     int *restrict);
14647 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
14648     int *restrict);
14649 int posix_trace_attr_getlogsize(const trace_attr_t *restrict,
14650     size_t *restrict);
14651 int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
14652     size_t *restrict);
14653 int posix_trace_attr_getmaxsystemeventsize(const trace_attr_t *restrict,
14654     size_t *restrict);
14655 int posix_trace_attr_getmaxusereventsize(const trace_attr_t *restrict,
14656     size_t, size_t *restrict);
14657 int posix_trace_attr_getname(const trace_attr_t *, char *);
14658 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
14659     int *restrict);
14660 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
14661     size_t *restrict);
14662 int posix_trace_attr_init(trace_attr_t *);
14663 TRI int posix_trace_attr_setinherited(trace_attr_t *, int);
14664 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
14665 int posix_trace_attr_setlogsize(trace_attr_t *, size_t);
14666 int posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
14667 int posix_trace_attr_setname(trace_attr_t *, const char *);
14668 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
14669 int posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
14670 int posix_trace_clear(trace_id_t);
14671 TRL int posix_trace_close(trace_id_t);
14672 int posix_trace_create(pid_t, const trace_attr_t *restrict,
14673     trace_id_t *restrict);
14674 TRL int posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
14675     int, trace_id_t *restrict);
14676 void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
14677 int posix_trace_eventid_equal(trace_id_t, trace_event_id_t,
14678     trace_event_id_t);
14679 int posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
14680 int posix_trace_eventid_open(const char *restrict,

```

```

14681         trace_event_id_t *restrict);
14682 TEF int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
14683 int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
14684 int  posix_trace_eventset_empty(trace_event_set_t *);
14685 int  posix_trace_eventset_fill(trace_event_set_t *, int);
14686 int  posix_trace_eventset_ismember(trace_event_id_t,
14687         const trace_event_set_t *restrict, int *restrict);
14688 int  posix_trace_eventtypelist_getnext_id(trace_id_t,
14689         trace_event_id_t *restrict, int *restrict);
14690 int  posix_trace_eventtypelist_rewind(trace_id_t);
14691 TRL int  posix_trace_flush(trace_id_t);
14692 int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
14693 TEF int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
14694 int  posix_trace_get_status(trace_id_t,
14695         struct posix_trace_status_info *);
14696 int  posix_trace_getnext_event(trace_id_t,
14697         struct posix_trace_event_info *restrict, void *restrict,
14698         size_t, size_t *restrict, int *restrict);
14699 TRL int  posix_trace_open(int, trace_id_t *);
14700 int  posix_trace_rewind(trace_id_t);
14701 TEF int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14702 int  posix_trace_shutdown(trace_id_t);
14703 int  posix_trace_start(trace_id_t);
14704 int  posix_trace_stop(trace_id_t);
14705 int  posix_trace_timedgetnext_event(trace_id_t,
14706         struct posix_trace_event_info *restrict, void *restrict,
14707         size_t, size_t *restrict, int *restrict,
14708         const struct timespec *restrict);
14709 TEF int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
14710         trace_event_id_t *restrict);
14711 int  posix_trace_trygetnext_event(trace_id_t,
14712         struct posix_trace_event_info *restrict, void *restrict, size_t,
14713         size_t *restrict, int *restrict);

```

14714 **APPLICATION USAGE**

14715 None.

14716 **RATIONALE**

14717 None.

14718 **FUTURE DIRECTIONS**

14719 The <trace.h> header may be removed in a future version.

14720 **SEE ALSO**

14721 [<sys/types.h>](#)

14722 XSH Section 2.11 (on page 536), [posix_trace_attr_destroy\(\)](#), [posix_trace_attr_getclockres\(\)](#),
14723 [posix_trace_attr_getinherited\(\)](#), [posix_trace_attr_getlogsize\(\)](#), [posix_trace_clear\(\)](#), [posix_trace_close\(\)](#),
14724 [posix_trace_create\(\)](#), [posix_trace_event\(\)](#), [posix_trace_eventid_equal\(\)](#), [posix_trace_eventset_add\(\)](#),
14725 [posix_trace_eventtypelist_getnext_id\(\)](#), [posix_trace_get_attr\(\)](#), [posix_trace_get_filter\(\)](#),
14726 [posix_trace_getnext_event\(\)](#), [posix_trace_start\(\)](#)

14727 **CHANGE HISTORY**

14728 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

14729 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/40 is applied, adding the TRL margin
14730 code to the *posix_trace_flush()* function, for alignment with the System Interfaces volume of
14731 POSIX.1-2017.

14732 **Issue 7**

14733 SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

14734 The **<trace.h>** header is marked obsolescent.

14735 This reference page is clarified with respect to macros and symbolic constants.

14736 **NAME**14737 `ulimit.h` ‡ulimit commands14738 **SYNOPSIS**14739 OB XSI `#include <ulimit.h>`14740 **DESCRIPTION**14741 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

14742 Symbolic constants:

14743 `UL_GETFSIZE` Get maximum file size.14744 `UL_SETFSIZE` Set maximum file size.14745 The following shall be declared as a function and may also be defined as a macro. A function
14746 prototype shall be provided.14747 `long ulimit(int, ...);`14748 **APPLICATION USAGE**14749 See *ulimit()*.14750 **RATIONALE**

14751 None.

14752 **FUTURE DIRECTIONS**

14753 None.

14754 **SEE ALSO**14755 XSH *ulimit()*14756 **CHANGE HISTORY**

14757 First released in Issue 3.

14758 **Issue 7**

14759 The <ulimit.h> header is marked obsolescent.

14760 **NAME**

14761 unistd.h †'standard symbolic constants and types

14762 **SYNOPSIS**

14763 #include <unistd.h>

14764 **DESCRIPTION**

14765 The **<unistd.h>** header defines miscellaneous symbolic constants and types, and declares
 14766 miscellaneous functions. The actual values of the constants are unspecified except as shown. The
 14767 contents of this header are shown below.

14768 **Version Test Macros**

14769 The **<unistd.h>** header shall define the following symbolic constants. The values shall be
 14770 suitable for use in **#if** preprocessing directives.

14771 **_POSIX_VERSION**

14772 Integer value indicating version of this standard (C-language binding) to which the
 14773 implementation conforms. For implementations conforming to POSIX.1-2017, the value
 14774 shall be 200809L.

14775 **_POSIX2_VERSION**

14776 Integer value indicating version of the Shell and Utilities volume of POSIX.1 to which the
 14777 implementation conforms. For implementations conforming to POSIX.1-2017, the value
 14778 shall be 200809L. For profile implementations that define **_POSIX_SUBPROFILE** (see
 14779 [Section 2.1.5.1](#)) in **<unistd.h>**, **_POSIX2_VERSION** may be left undefined or be defined with
 14780 the value **-1** to indicate that the Shell and Utilities volume of POSIX.1 is not supported. In
 14781 this case, a call to *sysconf(_SC_2_VERSION)* shall return either 200809L or **-1** indicating that
 14782 the Shell and Utilities volume of POSIX.1 is or is not, respectively, supported at runtime.

14783 The **<unistd.h>** header shall define the following symbolic constant only if the implementation
 14784 supports the XSI option; see [Section 2.1.4](#) (on page 19). If defined, its value shall be suitable for
 14785 use in **#if** preprocessing directives.

14786 XSI **_XOPEN_VERSION**

14787 Integer value indicating version of the X/Open Portability Guide to which the
 14788 implementation conforms. The value shall be 700.

14789 **Constants for Options and Option Groups**

14790 The following symbolic constants, if defined in **<unistd.h>**, shall have a value of **-1**, **0**, or
 14791 greater, unless otherwise specified below. For profile implementations that define
 14792 **_POSIX_SUBPROFILE** (see [Section 2.1.5.1](#)) in **<unistd.h>**, constants described below as always
 14793 having a value greater than zero need not be defined and, if defined, may have a value of **-1**, **0**,
 14794 or greater. The values shall be suitable for use in **#if** preprocessing directives.

14795 If a symbolic constant is not defined or is defined with the value **-1**, the option is not supported
 14796 for compilation. If it is defined with a value greater than zero, the option shall always be
 14797 supported when the application is executed. If it is defined with the value zero, the option shall
 14798 be supported for compilation and might or might not be supported at runtime. See [Section 2.1.6](#)
 14799 (on page 26) for further information about the conformance requirements of these three
 14800 categories of support.

14801 ADV **_POSIX_ADVISORY_INFO**

14802 The implementation supports the Advisory Information option. If this symbol is defined in
 14803 **<unistd.h>**, it shall be defined to be **-1**, **0**, or 200809L. The value of this symbol reported by
 14804 *sysconf()* shall either be **-1** or 200809L.

14805 `_POSIX_ASYNCHRONOUS_IO`
 14806 The implementation supports asynchronous input and output. This symbol shall always be
 14807 set to the value 200809L.

14808 `_POSIX_BARRIERS`
 14809 The implementation supports barriers. This symbol shall always be set to the value
 14810 200809L.

14811 `_POSIX_CHOWN_RESTRICTED`
 14812 The use of *chown()* and *fchown()* is restricted to a process with appropriate privileges, and
 14813 to changing the group ID of a file only to the effective group ID of the process or to one of
 14814 its supplementary group IDs. This symbol shall be defined with a value other than -1.

14815 `_POSIX_CLOCK_SELECTION`
 14816 The implementation supports clock selection. This symbol shall always be set to the value
 14817 200809L.

14818 CPT `_POSIX_CPUTIME`
 14819 The implementation supports the Process CPU-Time Clocks option. If this symbol is
 14820 defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
 14821 reported by *sysconf()* shall either be -1 or 200809L.

14822 FSC `_POSIX_FSYNC`
 14823 The implementation supports the File Synchronization option. If this symbol is defined in
 14824 <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
 14825 *sysconf()* shall either be -1 or 200809L.

14826 IP6 `_POSIX_IPV6`
 14827 The implementation supports the IPv6 option. If this symbol is defined in <unistd.h>, it
 14828 shall be defined to be -1, 0, or 200809L. The value of this symbol reported by *sysconf()* shall
 14829 either be -1 or 200809L.

14830 `_POSIX_JOB_CONTROL`
 14831 The implementation supports job control. This symbol shall always be set to a value greater
 14832 than zero.

14833 `_POSIX_MAPPED_FILES`
 14834 The implementation supports memory mapped Files. This symbol shall always be set to the
 14835 value 200809L.

14836 ML `_POSIX_MEMLOCK`
 14837 The implementation supports the Process Memory Locking option. If this symbol is defined
 14838 in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
 14839 by *sysconf()* shall either be -1 or 200809L.

14840 MLR `_POSIX_MEMLOCK_RANGE`
 14841 The implementation supports the Range Memory Locking option. If this symbol is defined
 14842 in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
 14843 by *sysconf()* shall either be -1 or 200809L.

14844 `_POSIX_MEMORY_PROTECTION`
 14845 The implementation supports memory protection. This symbol shall always be set to the
 14846 value 200809L.

14847 MSG `_POSIX_MESSAGE_PASSING`
 14848 The implementation supports the Message Passing option. If this symbol is defined in
 14849 <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
 14850 *sysconf()* shall either be -1 or 200809L.

14851	MON	_POSIX_MONOTONIC_CLOCK
14852		The implementation supports the Monotonic Clock option. If this symbol is defined in
14853		<unistd.h> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14854		<i>sysconf()</i> shall either be <code>-1</code> or <code>200809L</code> .
14855		_POSIX_NO_TRUNC
14856		Pathname components longer than <code>{NAME_MAX}</code> generate an error. This symbol shall be
14857		defined with a value other than <code>-1</code> .
14858	PIO	_POSIX_PRIORITIZED_IO
14859		The implementation supports the Prioritized Input and Output option. If this symbol is
14860		defined in <unistd.h> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol
14861		reported by <i>sysconf()</i> shall either be <code>-1</code> or <code>200809L</code> .
14862	PS	_POSIX_PRIORITY_SCHEDULING
14863		The implementation supports the Process Scheduling option. If this symbol is defined in
14864		<unistd.h> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14865		<i>sysconf()</i> shall either be <code>-1</code> or <code>200809L</code> .
14866	RS	_POSIX_RAW_SOCKETS
14867		The implementation supports the Raw Sockets option. If this symbol is defined in
14868		<unistd.h> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by
14869		<i>sysconf()</i> shall either be <code>-1</code> or <code>200809L</code> .
14870		_POSIX_READER_WRITER_LOCKS
14871		The implementation supports read-write locks. This symbol shall always be set to the value
14872		<code>200809L</code> .
14873		_POSIX_REALTIME_SIGNALS
14874		The implementation supports realtime signals. This symbol shall always be set to the value
14875		<code>200809L</code> .
14876		_POSIX_REGEX
14877		The implementation supports the Regular Expression Handling option. This symbol shall
14878		always be set to a value greater than zero.
14879		_POSIX_SAVED_IDS
14880		Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be
14881		set to a value greater than zero.
14882		_POSIX_SEMAPHORES
14883		The implementation supports semaphores. This symbol shall always be set to the value
14884		<code>200809L</code> .
14885	SHM	_POSIX_SHARED_MEMORY_OBJECTS
14886		The implementation supports the Shared Memory Objects option. If this symbol is defined
14887		in <unistd.h> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported
14888		by <i>sysconf()</i> shall either be <code>-1</code> or <code>200809L</code> .
14889		_POSIX_SHELL
14890		The implementation supports the POSIX shell. This symbol shall always be set to a value
14891		greater than zero.
14892	SPN	_POSIX_SPAWN
14893		The implementation supports the Spawn option. If this symbol is defined in <unistd.h> , it
14894		shall be defined to be <code>-1</code> , <code>0</code> , or <code>200809L</code> . The value of this symbol reported by <i>sysconf()</i> shall
14895		either be <code>-1</code> or <code>200809L</code> .

14896		_POSIX_SPIN_LOCKS
14897		The implementation supports spin locks. This symbol shall always be set to the value
14898		200809L.
14899	SS	_POSIX_SPORADIC_SERVER
14900		The implementation supports the Process Sporadic Server option. If this symbol is defined
14901		in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
14902		by <i>sysconf()</i> shall either be -1 or 200809L.
14903	SIO	_POSIX_SYNCHRONIZED_IO
14904		The implementation supports the Synchronized Input and Output option. If this symbol is
14905		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14906		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14907	TSA	_POSIX_THREAD_ATTR_STACKADDR
14908		The implementation supports the Thread Stack Address Attribute option. If this symbol is
14909		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14910		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14911	TSS	_POSIX_THREAD_ATTR_STACKSIZE
14912		The implementation supports the Thread Stack Size Attribute option. If this symbol is
14913		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14914		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14915	TCT	_POSIX_THREAD_CPU_TIME
14916		The implementation supports the Thread CPU-Time Clocks option. If this symbol is
14917		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14918		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14919	TPI	_POSIX_THREAD_PRIO_INHERIT
14920		The implementation supports the Non-Robust Mutex Priority Inheritance option. If this
14921		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this
14922		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14923	TPP	_POSIX_THREAD_PRIO_PROTECT
14924		The implementation supports the Non-Robust Mutex Priority Protection option. If this
14925		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this
14926		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14927	TPS	_POSIX_THREAD_PRIORITY_SCHEDULING
14928		The implementation supports the Thread Execution Scheduling option. If this symbol is
14929		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14930		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14931	TSH	_POSIX_THREAD_PROCESS_SHARED
14932		The implementation supports the Thread Process-Shared Synchronization option. If this
14933		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this
14934		symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
14935	RPI	_POSIX_THREAD_ROBUST_PRIO_INHERIT
14936		The implementation supports the Robust Mutex Priority Inheritance option. If this symbol
14937		is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol
14938		reported by <i>sysconf()</i> shall either be -1 or 200809L.
14939	RPP	_POSIX_THREAD_ROBUST_PRIO_PROTECT
14940		The implementation supports the Robust Mutex Priority Protection option. If this symbol is
14941		defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol

- 14942 reported by *sysconf()* shall either be `-1` or `200809L`.
- 14943 `_POSIX_THREAD_SAFE_FUNCTIONS`
14944 The implementation supports thread-safe functions. This symbol shall always be set to the
14945 value `200809L`.
- 14946 TSP `_POSIX_THREAD_SPORADIC_SERVER`
14947 The implementation supports the Thread Sporadic Server option. If this symbol is defined
14948 in `<unistd.h>`, it shall be defined to be `-1`, `0`, or `200809L`. The value of this symbol reported
14949 by *sysconf()* shall either be `-1` or `200809L`.
- 14950 `_POSIX_THREADS`
14951 The implementation supports threads. This symbol shall always be set to the value
14952 `200809L`.
- 14953 `_POSIX_TIMEOUTS`
14954 The implementation supports timeouts. This symbol shall always be set to the value
14955 `200809L`.
- 14956 `_POSIX_TIMERS`
14957 The implementation supports timers. This symbol shall always be set to the value `200809L`.
- 14958 OB TRC `_POSIX_TRACE`
14959 The implementation supports the Trace option. If this symbol is defined in `<unistd.h>`, it
14960 shall be defined to be `-1`, `0`, or `200809L`. The value of this symbol reported by *sysconf()* shall
14961 either be `-1` or `200809L`.
- 14962 OB TEF `_POSIX_TRACE_EVENT_FILTER`
14963 The implementation supports the Trace Event Filter option. If this symbol is defined in
14964 `<unistd.h>`, it shall be defined to be `-1`, `0`, or `200809L`. The value of this symbol reported by
14965 *sysconf()* shall either be `-1` or `200809L`.
- 14966 OB TRI `_POSIX_TRACE_INHERIT`
14967 The implementation supports the Trace Inherit option. If this symbol is defined in
14968 `<unistd.h>`, it shall be defined to be `-1`, `0`, or `200809L`. The value of this symbol reported by
14969 *sysconf()* shall either be `-1` or `200809L`.
- 14970 OB TRL `_POSIX_TRACE_LOG`
14971 The implementation supports the Trace Log option. If this symbol is defined in `<unistd.h>`,
14972 it shall be defined to be `-1`, `0`, or `200809L`. The value of this symbol reported by *sysconf()*
14973 shall either be `-1` or `200809L`.
- 14974 TYM `_POSIX_TYPED_MEMORY_OBJECTS`
14975 The implementation supports the Typed Memory Objects option. If this symbol is defined
14976 in `<unistd.h>`, it shall be defined to be `-1`, `0`, or `200809L`. The value of this symbol reported
14977 by *sysconf()* shall either be `-1` or `200809L`.
- 14978 OB `_POSIX_V6_ILP32_OFF32`
14979 The implementation provides a C-language compilation environment with 32-bit **int**, **long**,
14980 **pointer**, and **off_t** types.
- 14981 OB `_POSIX_V6_ILP32_OFFBIG`
14982 The implementation provides a C-language compilation environment with 32-bit **int**, **long**,
14983 and **pointer** types and an **off_t** type using at least 64 bits.
- 14984 OB `_POSIX_V6_LP64_OFF64`
14985 The implementation provides a C-language compilation environment with 32-bit **int** and
14986 64-bit **long**, **pointer**, and **off_t** types.

14987	OB	_POSIX_V6_LPBIG_OFFBIG	The implementation provides a C-language compilation environment with an int type using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14988			
14989			
14990		_POSIX_V7_ILP32_OFF32	The implementation provides a C-language compilation environment with 32-bit int , long , pointer , and off_t types.
14991			
14992			
14993		_POSIX_V7_ILP32_OFFBIG	The implementation provides a C-language compilation environment with 32-bit int , long , and pointer types and an off_t type using at least 64 bits.
14994			
14995			
14996		_POSIX_V7_LP64_OFF64	The implementation provides a C-language compilation environment with 32-bit int and 64-bit long , pointer , and off_t types.
14997			
14998			
14999		_POSIX_V7_LPBIG_OFFBIG	The implementation provides a C-language compilation environment with an int type using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
15000			
15001			
15002		_POSIX2_C_BIND	The implementation supports the C-Language Binding option. This symbol shall always have the value 200809L.
15003			
15004			
15005	CD	_POSIX2_C_DEV	The implementation supports the C-Language Development Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
15006			
15007			
15008			
15009		_POSIX2_CHAR_TERM	The implementation supports the Terminal Characteristics option. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or a value greater than zero.
15010			
15011			
15012	FD	_POSIX2_FORT_DEV	The implementation supports the FORTRAN Development Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
15013			
15014			
15015			
15016	FR	_POSIX2_FORT_RUN	The implementation supports the FORTRAN Runtime Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
15017			
15018			
15019			
15020		_POSIX2_LOCALEDEF	The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
15021			
15022			
15023			
15024	OB BE	_POSIX2_PBS	The implementation supports the Batch Environment Services and Utilities option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
15025			
15026			
15027			
15028	OB BE	_POSIX2_PBS_ACCOUNTING	The implementation supports the Batch Accounting option. If this symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by <i>sysconf()</i> shall either be -1 or 200809L.
15029			
15030			
15031			

15032	OB BE	<u>POSIX2_PBS_CHECKPOINT</u>
15033		The implementation supports the Batch Checkpoint/Restart option. If this symbol is
15034		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
15035		reported by <i>sysconf()</i> shall either be -1 or 200809L.
15036	OB BE	<u>POSIX2_PBS_LOCATE</u>
15037		The implementation supports the Locate Batch Job Request option. If this symbol is defined
15038		in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
15039		by <i>sysconf()</i> shall either be -1 or 200809L.
15040	OB BE	<u>POSIX2_PBS_MESSAGE</u>
15041		The implementation supports the Batch Job Message Request option. If this symbol is
15042		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
15043		reported by <i>sysconf()</i> shall either be -1 or 200809L.
15044	OB BE	<u>POSIX2_PBS_TRACK</u>
15045		The implementation supports the Track Batch Job Request option. If this symbol is defined
15046		in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
15047		by <i>sysconf()</i> shall either be -1 or 200809L.
15048	SD	<u>POSIX2_SW_DEV</u>
15049		The implementation supports the Software Development Utilities option. If this symbol is
15050		defined in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol
15051		reported by <i>sysconf()</i> shall either be -1 or 200809L.
15052	UP	<u>POSIX2_UPE</u>
15053		The implementation supports the User Portability Utilities option. If this symbol is defined
15054		in <unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol reported
15055		by <i>sysconf()</i> shall either be -1 or 200809L.
15056	XSI	<u>XOPEN_CRYPT</u>
15057		The implementation supports the X/Open Encryption Option Group.
15058		<u>XOPEN_ENH_I18N</u>
15059		The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
15060		Group. This symbol shall always be set to a value other than -1.
15061		<u>XOPEN_REALTIME</u>
15062		The implementation supports the X/Open Realtime Option Group.
15063		<u>XOPEN_REALTIME_THREADS</u>
15064		The implementation supports the X/Open Realtime Threads Option Group.
15065		<u>XOPEN_SHM</u>
15066		The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
15067		symbol shall always be set to a value other than -1.
15068	OB XSR	<u>XOPEN_STREAMS</u>
15069		The implementation supports the XSI STREAMS Option Group.
15070	XSI	<u>XOPEN_UNIX</u>
15071		The implementation supports the XSI option.
15072	UU	<u>XOPEN_UUCP</u>
15073		The implementation supports the UUCP Utilities option. If this symbol is defined in
15074		<unistd.h> , it shall be defined to be -1, 0, or 200809L. The value of this symbol reported by
15075		<i>sysconf()</i> shall be either -1 or 200809L.

15076 **Execution-Time Symbolic Constants**

15077 If any of the following symbolic constants are not defined in the <unistd.h> header, the value
 15078 shall vary depending on the file to which it is applied. If defined, they shall have values suitable
 15079 for use in **#if** preprocessing directives.

15080 If any of the following symbolic constants are defined to have value `-1` in the <unistd.h> header,
 15081 the implementation shall not provide the option on any file; if any are defined to have a value
 15082 other than `-1` in the <unistd.h> header, the implementation shall provide the option on all
 15083 applicable files.

15084 All of the following values, whether defined as symbolic constants in <unistd.h> or not, may be
 15085 queried with respect to a specific file using the *pathconf()* or *fpathconf()* functions:

15086 `_POSIX_ASYNC_IO`

15087 Asynchronous input or output operations may be performed for the associated file.

15088 `_POSIX_PRIO_IO`

15089 Prioritized input or output operations may be performed for the associated file.

15090 `_POSIX_SYNC_IO`

15091 Synchronized input or output operations may be performed for the associated file.

15092 If the following symbolic constants are defined in the <unistd.h> header, they apply to files and
 15093 all paths in all file systems on the implementation:

15094 `_POSIX_TIMESTAMP_RESOLUTION`

15095 The resolution in nanoseconds for all file timestamps.

15096 `_POSIX2_SYMLINKS`

15097 Symbolic links can be created.

15098 **Constants for Functions**

15099 The <unistd.h> header shall define `NULL` as described in <stddef.h>.

15100 The <unistd.h> header shall define the following symbolic constants for use with the *access()*
 15101 function. The values shall be suitable for use in **#if** preprocessing directives.

15102 `F_OK` Test for existence of file.

15103 `R_OK` Test for read permission.

15104 `W_OK` Test for write permission.

15105 `X_OK` Test for execute (search) permission.

15106 The constants `F_OK`, `R_OK`, `W_OK`, and `X_OK` and the expressions `R_OK | W_OK`, `R_OK | X_OK`,
 15107 and `R_OK | W_OK | X_OK` shall all have distinct values.

15108 The <unistd.h> header shall define the following symbolic constants for the *confstr()* function:

15109 `_CS_PATH`

15110 This is the value for the *PATH* environment variable that finds all of the standard utilities
 15111 that are provided in a manner accessible via the *exec* family of functions.

15112 `_CS_POSIX_V7_ILP32_OFF32_CFLAGS`

15113 If *sysconf(_SC_V7_ILP32_OFF32)* returns `-1`, the meaning of this value is unspecified.
 15114 Otherwise, this value is the set of initial options to be given to the *c99* utility to build an
 15115 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15116 `_CS_POSIX_V7_ILP32_OFF32_LDFLAGS`
15117 If `sysconf(_SC_V7_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15118 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
15119 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15120 `_CS_POSIX_V7_ILP32_OFF32_LIBS`
15121 If `sysconf(_SC_V7_ILP32_OFF32)` returns `-1`, the meaning of this value is unspecified.
15122 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
15123 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

15124 `_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS`
15125 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15126 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
15127 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15128 **off_t** type using at least 64 bits.

15129 `_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS`
15130 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15131 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
15132 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15133 **off_t** type using at least 64 bits.

15134 `_CS_POSIX_V7_ILP32_OFFBIG_LIBS`
15135 If `sysconf(_SC_V7_ILP32_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15136 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
15137 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
15138 **off_t** type using at least 64 bits.

15139 `_CS_POSIX_V7_LP64_OFF64_CFLAGS`
15140 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
15141 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
15142 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
15143 types.

15144 `_CS_POSIX_V7_LP64_OFF64_LDFLAGS`
15145 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
15146 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
15147 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
15148 types.

15149 `_CS_POSIX_V7_LP64_OFF64_LIBS`
15150 If `sysconf(_SC_V7_LP64_OFF64)` returns `-1`, the meaning of this value is unspecified.
15151 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
15152 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
15153 types.

15154 `_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS`
15155 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15156 Otherwise, this value is the set of initial options to be given to the `c99` utility to build an
15157 application using a programming model with an **int** type using at least 32 bits and **long**,
15158 **pointer**, and **off_t** types using at least 64 bits.

15159 `_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS`
15160 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
15161 Otherwise, this value is the set of final options to be given to the `c99` utility to build an
15162 application using a programming model with an **int** type using at least 32 bits and **long**,

15163 **pointer**, and **off_t** types using at least 64 bits.

15164 `_CS_POSIX_V7_LPBIG_OFFBIG_LIBS`
 15165 If `sysconf(_SC_V7_LPBIG_OFFBIG)` returns `-1`, the meaning of this value is unspecified.
 15166 Otherwise, this value is the set of libraries to be given to the `c99` utility to build an
 15167 application using a programming model with an **int** type using at least 32 bits and **long**,
 15168 **pointer**, and **off_t** types using at least 64 bits.

15169 `_CS_POSIX_V7_THREADS_CFLAGS`
 15170 If `sysconf(_SC_POSIX_THREADS)` returns `-1`, the meaning of this value is unspecified.
 15171 Otherwise, this value is the set of initial options to be given to the `c99` utility to build a
 15172 multi-threaded application. These flags are in addition to those associated with any of the
 15173 other `_CS_POSIX_V7_*_CFLAGS` values used to specify particular type size programing
 15174 environments.

15175 `_CS_POSIX_V7_THREADS_LDFLAGS`
 15176 If `sysconf(_SC_POSIX_THREADS)` returns `-1`, the meaning of this value is unspecified.
 15177 Otherwise, this value is the set of final options to be given to the `c99` utility to build a multi-
 15178 threaded application. These flags are in addition to those associated with any of the other
 15179 `_CS_POSIX_V7*_LDFLAGS` values used to specify particular type size programing
 15180 environments.

15181 `_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS`
 15182 This value is a <newline>-separated list of names of programming environments supported
 15183 by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**,
 15184 **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **wchar_t**, and **wint_t** types are no
 15185 greater than the width of type **long**. The format of each name shall be suitable for use with
 15186 the `getconf -v` option.

15187 `_CS_V7_ENV`
 15188 This is the value that provides the environment variable information (other than that
 15189 provided by `_CS_PATH`) that is required by the implementation to create a conforming
 15190 environment, as described in the implementation's conformance documentation.

15191 OB The following symbolic constants are reserved for compatibility with Issue 6:

15192 `_CS_POSIX_V6_ILP32_OFF32_CFLAGS`
 15193 `_CS_POSIX_V6_ILP32_OFF32_LDFLAGS`
 15194 `_CS_POSIX_V6_ILP32_OFF32_LIBS`
 15195 `_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS`
 15196 `_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS`
 15197 `_CS_POSIX_V6_ILP32_OFFBIG_LIBS`
 15198 `_CS_POSIX_V6_LP64_OFF64_CFLAGS`
 15199 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`
 15200 `_CS_POSIX_V6_LP64_OFF64_LIBS`
 15201 `_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS`
 15202 `_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS`
 15203 `_CS_POSIX_V6_LPBIG_OFFBIG_LIBS`
 15204 `_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS`
 15205 `_CS_V6_ENV`

15206 The <unistd.h> header shall define `SEEK_CUR`, `SEEK_END`, and `SEEK_SET` as described in
 15207 <stdio.h>.

15208 XSI The **<unistd.h>** header shall define the following symbolic constants as possible values for the
15209 *function* argument to the *lockf()* function:

15210 **F_LOCK** Lock a section for exclusive use.
15211 **F_TEST** Test section for locks by other processes.
15212 **F_TLOCK** Test and lock a section for exclusive use.
15213 **F_ULOCK** Unlock locked sections.

15214 The **<unistd.h>** header shall define the following symbolic constants for *pathconf()*:

15215 **_PC_2_SYMLINKS**
15216 **_PC_ALLOC_SIZE_MIN**
15217 **_PC_ASYNC_IO**
15218 **_PC_CHOWN_RESTRICTED**
15219 **_PC_FILESIZEBITS**
15220 **_PC_LINK_MAX**
15221 **_PC_MAX_CANON**
15222 **_PC_MAX_INPUT**
15223 **_PC_NAME_MAX**
15224 **_PC_NO_TRUNC**
15225 **_PC_PATH_MAX**
15226 **_PC_PIPE_BUF**
15227 **_PC_PRIO_IO**
15228 **_PC_REC_INCR_XFER_SIZE**
15229 **_PC_REC_MAX_XFER_SIZE**
15230 **_PC_REC_MIN_XFER_SIZE**
15231 **_PC_REC_XFER_ALIGN**
15232 **_PC_SYMLINK_MAX**
15233 **_PC_SYNC_IO**
15234 **_PC_TIMESTAMP_RESOLUTION**
15235 **_PC_VDISABLE**

15236 The **<unistd.h>** header shall define the following symbolic constants for *sysconf()*:

15237 **_SC_2_C_BIND**
15238 **_SC_2_C_DEV**
15239 **_SC_2_CHAR_TERM**
15240 **_SC_2_FORT_DEV**
15241 **_SC_2_FORT_RUN**
15242 **_SC_2_LOCALEDEF**
15243 **_SC_2_PBS**
15244 **_SC_2_PBS_ACCOUNTING**
15245 **_SC_2_PBS_CHECKPOINT**
15246 **_SC_2_PBS_LOCATE**
15247 **_SC_2_PBS_MESSAGE**
15248 **_SC_2_PBS_TRACK**
15249 **_SC_2_SW_DEV**
15250 **_SC_2_UPE**
15251 **_SC_2_VERSION**
15252 **_SC_ADVISORY_INFO**
15253 **_SC_AIO_LISTIO_MAX**
15254 **_SC_AIO_MAX**

15255 _SC_AIO_PRIO_DELTA_MAX
15256 _SC_ARG_MAX
15257 _SC_ASYNCCHRONOUS_IO
15258 _SC_ATEXIT_MAX
15259 _SC_BARRIERS
15260 _SC_BC_BASE_MAX
15261 _SC_BC_DIM_MAX
15262 _SC_BC_SCALE_MAX
15263 _SC_BC_STRING_MAX
15264 _SC_CHILD_MAX
15265 _SC_CLK_TCK
15266 _SC_CLOCK_SELECTION
15267 _SC_COLL_WEIGHTS_MAX
15268 _SC_CPUTIME
15269 _SC_DELAYTIMER_MAX
15270 _SC_EXPR_NEST_MAX
15271 _SC_FSYNC
15272 _SC_GETGR_R_SIZE_MAX
15273 _SC_GETPW_R_SIZE_MAX
15274 _SC_HOST_NAME_MAX
15275 _SC_IOV_MAX
15276 _SC_IPV6
15277 _SC_JOB_CONTROL
15278 _SC_LINE_MAX
15279 _SC_LOGIN_NAME_MAX
15280 _SC_MAPPED_FILES
15281 _SC_MEMLOCK
15282 _SC_MEMLOCK_RANGE
15283 _SC_MEMORY_PROTECTION
15284 _SC_MESSAGE_PASSING
15285 _SC_MONOTONIC_CLOCK
15286 _SC_MQ_OPEN_MAX
15287 _SC_MQ_PRIO_MAX
15288 _SC_NGROUPS_MAX
15289 _SC_OPEN_MAX
15290 _SC_PAGE_SIZE
15291 _SC_PAGESIZE
15292 _SC_PRIORITIZED_IO
15293 _SC_PRIORITY_SCHEDULING
15294 _SC_RAW_SOCKETS
15295 _SC_RE_DUP_MAX
15296 _SC_READER_WRITER_LOCKS
15297 _SC_REALTIME_SIGNALS
15298 _SC_REGEX
15299 _SC_RT_SIG_MAX
15300 _SC_SAVED_IDS
15301 _SC_SEM_NSEMS_MAX
15302 _SC_SEM_VALUE_MAX
15303 _SC_SEMAPHORES
15304 _SC_SHARED_MEMORY_OBJECTS
15305 _SC_SHELL
15306 _SC_SIGQUEUE_MAX

15307 _SC_SPAWN
15308 _SC_SPIN_LOCKS
15309 _SC_SPORADIC_SERVER
15310 _SC_SS_REPL_MAX
15311 _SC_STREAM_MAX
15312 _SC_SYMLoop_MAX
15313 _SC_SYNCHRONIZED_IO
15314 _SC_THREAD_ATTR_STACKADDR
15315 _SC_THREAD_ATTR_STACKSIZE
15316 _SC_THREAD_CPUTIME
15317 _SC_THREAD_DESTRUCTOR_ITERATIONS
15318 _SC_THREAD_KEYS_MAX
15319 _SC_THREAD_PRIO_INHERIT
15320 _SC_THREAD_PRIO_PROTECT
15321 _SC_THREAD_PRIORITY_SCHEDULING
15322 _SC_THREAD_PROCESS_SHARED
15323 _SC_THREAD_ROBUST_PRIO_INHERIT
15324 _SC_THREAD_ROBUST_PRIO_PROTECT
15325 _SC_THREAD_SAFE_FUNCTIONS
15326 _SC_THREAD_SPORADIC_SERVER
15327 _SC_THREAD_STACK_MIN
15328 _SC_THREAD_THREADS_MAX
15329 _SC_THREADS
15330 _SC_TIMEOUTS
15331 _SC_TIMER_MAX
15332 _SC_TIMERS
15333 _SC_TRACE
15334 _SC_TRACE_EVENT_FILTER
15335 _SC_TRACE_EVENT_NAME_MAX
15336 _SC_TRACE_INHERIT
15337 _SC_TRACE_LOG
15338 _SC_TRACE_NAME_MAX
15339 _SC_TRACE_SYS_MAX
15340 _SC_TRACE_USER_EVENT_MAX
15341 _SC_TTY_NAME_MAX
15342 _SC_TYPED_MEMORY_OBJECTS
15343 _SC_TZNAME_MAX
15344 _SC_V7_ILP32_OFF32
15345 _SC_V7_ILP32_OFFBIG
15346 _SC_V7_LP64_OFF64
15347 _SC_V7_LPBIG_OFFBIG
15348 OB _SC_V6_ILP32_OFF32
15349 _SC_V6_ILP32_OFFBIG
15350 _SC_V6_LP64_OFF64
15351 _SC_V6_LPBIG_OFFBIG
15352 _SC_VERSION
15353 _SC_XOPEN_CRYPT
15354 _SC_XOPEN_ENH_I18N
15355 _SC_XOPEN_REALTIME
15356 _SC_XOPEN_REALTIME_THREADS
15357 _SC_XOPEN_SHM
15358 _SC_XOPEN_STREAMS

15359 _SC_XOPEN_UNIX
 15360 _SC_XOPEN_UUCP
 15361 _SC_XOPEN_VERSION

15362 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same value.

15363 The <unistd.h> header shall define the following symbolic constants for file streams:

15364 STDERR_FILENO File number of *stderr*; 2.
 15365 STDIN_FILENO File number of *stdin*; 0.
 15366 STDOUT_FILENO File number of *stdout*; 1.

15367 The <unistd.h> header shall define the following symbolic constant for terminal special
 15368 character handling:

15369 _POSIX_VDISABLE This symbol shall be defined to be the value of a character that shall
 15370 disable terminal special character handling as described in [Section 11.2.6](#)
 15371 (on page 212). This symbol shall always be set to a value other than -1.

15372 **Type Definitions**

15373 The <unistd.h> header shall define the `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types as
 15374 described in <sys/types.h>.

15375 The <unistd.h> header shall define the `intptr_t` type as described in <stdint.h>.

15376 **Declarations**

15377 The following shall be declared as functions and may also be defined as macros. Function
 15378 prototypes shall be provided.

```

15379        int            access(const char *, int);
15380        unsigned       alarm(unsigned);
15381        int            chdir(const char *);
15382        int            chown(const char *, uid_t, gid_t);
15383        int            close(int);
15384        size_t         confstr(int, char *, size_t);
15385 XSI       char        *crypt(const char *, const char *);
15386        int            dup(int);
15387        int            dup2(int, int);
15388        void           _exit(int);
15389 XSI       void        encrypt(char [64], int);
15390        int            execl(const char *, const char *, ...);
15391        int            execle(const char *, const char *, ...);
15392        int            execlp(const char *, const char *, ...);
15393        int            execv(const char *, char *const []);
15394        int            execve(const char *, char *const [], char *const []);
15395        int            execvp(const char *, char *const []);
15396        int            faccessat(int, const char *, int, int);
15397        int            fchdir(int);
15398        int            fchown(int, uid_t, gid_t);
15399        int            fchownat(int, const char *, uid_t, gid_t, int);
15400 SIO       int        fdatasync(int);
15401        int            fexecve(int, char *const [], char *const []);
15402        pid_t          fork(void);
    
```

```

15403     long      fpathconf(int, int);
15404 FSC    int      fsync(int);
15405     int      ftruncate(int, off_t);
15406     char      *getcwd(char *, size_t);
15407     gid_t     getegid(void);
15408     uid_t     geteuid(void);
15409     gid_t     getgid(void);
15410     int      getgroups(int, gid_t []);
15411 XSI    long      gethostid(void);
15412     int      gethostname(char *, size_t);
15413     char      *getlogin(void);
15414     int      getlogin_r(char *, size_t);
15415     int      getopt(int, char * const [], const char *);
15416     pid_t     getpgid(pid_t);
15417     pid_t     getpgrp(void);
15418     pid_t     getpid(void);
15419     pid_t     getppid(void);
15420     pid_t     getsid(pid_t);
15421     uid_t     getuid(void);
15422     int      isatty(int);
15423     int      lchown(const char *, uid_t, gid_t);
15424     int      link(const char *, const char *);
15425     int      linkat(int, const char *, int, const char *, int);
15426 XSI    int      lockf(int, int, off_t);
15427     off_t     lseek(int, off_t, int);
15428 XSI    int      nice(int);
15429     long     pathconf(const char *, int);
15430     int      pause(void);
15431     int      pipe(int [2]);
15432     ssize_t   pread(int, void *, size_t, off_t);
15433     ssize_t   pwrite(int, const void *, size_t, off_t);
15434     ssize_t   read(int, void *, size_t);
15435     ssize_t   readlink(const char *restrict, char *restrict, size_t);
15436     ssize_t   readlinkat(int, const char *restrict, char *restrict, size_t);
15437     int      rmdir(const char *);
15438     int      setegid(gid_t);
15439     int      seteuid(uid_t);
15440     int      setgid(gid_t);
15441     int      setpgid(pid_t, pid_t);
15442 OB XSI  pid_t     setpgrp(void);
15443 XSI    int      setregid(gid_t, gid_t);
15444     int      setreuid(uid_t, uid_t);
15445     pid_t     setsid(void);
15446     int      setuid(uid_t);
15447     unsigned  sleep(unsigned);
15448 XSI    void     swab(const void *restrict, void *restrict, ssize_t);
15449     int      symlink(const char *, const char *);
15450     int      symlinkat(const char *, int, const char *);
15451 XSI    void     sync(void);
15452     long     sysconf(int);
15453     pid_t     tcgetpgrp(int);
15454     int      tcsetpgrp(int, pid_t);

```

```

15455     int           truncate(const char *, off_t);
15456     char          *ttyname(int);
15457     int           ttyname_r(int, char *, size_t);
15458     int           unlink(const char *);
15459     int           unlinkat(int, const char *, int);
15460     ssize_t       write(int, const void *, size_t);

```

15461 OB Implementations may also include the `pthread_atfork()` prototype as defined in `<pthread.h>`.
 15462 Implementations may also include the `ctermid()` prototype as defined in `<stdio.h>`.

15463 The `<unistd.h>` header shall declare the following external variables:

```

15464     extern char   *optarg;
15465     extern int    opterr, optind, optopt;

```

15466 Inclusion of the `<unistd.h>` header may make visible all symbols from the headers `<stddef.h>`,
 15467 `<stdint.h>`, and `<stdio.h>`.

15468 APPLICATION USAGE

15469 POSIX.1-2017 only describes the behavior of systems that claim conformance to it. However,
 15470 application developers who want to write applications that adapt to other versions of this
 15471 standard (or to systems that do not conform to any POSIX standard) may find it useful to code
 15472 them so as to conditionally compile different code depending on the value of
 15473 `_POSIX_VERSION`, for example:

```

15474     #if _POSIX_VERSION >= 200112L
15475     /* Use the newer function that copes with large files. */
15476     off_t pos=ftello(fp);
15477     #else
15478     /* Either this is an old version of POSIX, or _POSIX_VERSION is
15479        not even defined, so use the traditional function. */
15480     long pos=ftell(fp);
15481     #endif

```

15482 Earlier versions of POSIX.1-2017 and of the Single UNIX Specification can be identified by the
 15483 following macros:

15484 POSIX.1-1988 standard
 15485 `_POSIX_VERSION == 198808L`

15486 POSIX.1-1990 standard
 15487 `_POSIX_VERSION == 199009L`

15488 ISO POSIX-1: 1996 standard
 15489 `_POSIX_VERSION == 199506L`

15490 Single UNIX Specification, Version 1
 15491 `_XOPEN_UNIX and _XOPEN_VERSION == 4`

15492 Single UNIX Specification, Version 2
 15493 `_XOPEN_UNIX and _XOPEN_VERSION == 500`

15494 ISO POSIX-1: 2001 and Single UNIX Specification, Version 3
 15495 `_POSIX_VERSION == 200112L`, plus (if the XSI option is supported) `_XOPEN_UNIX` and
 15496 `_XOPEN_VERSION == 600`

15497 POSIX.1-2017 does not make any attempt to define application binary interaction with the
 15498 underlying operating system. However, application developers may find it useful to query
 15499 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the

15500 operating system supports the necessary functionality as in the following program fragment:

```
15501 if (sysconf(_SC_VERSION) < 200809L) {  
15502     fprintf(stderr, "POSIX.1-2008 system required, terminating \n");  
15503     exit(1);  
15504 }
```

15505 New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`.

15506 RATIONALE

15507 As POSIX.1-2017 evolved, certain options became sufficiently standardized that it was
15508 concluded that simply requiring one of the option choices was simpler than retaining the option.
15509 However, for backwards-compatibility, the option flags (with required constant values) are
15510 retained.

15511 Version Test Macros

15512 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
15513 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
15514 by some to be minimal, and since the implementation of the functionality is potentially
15515 problematic. However, they recognized that support for existing application binaries is a
15516 concern to manufacturers, application developers, and the users of implementations conforming
15517 to POSIX.1-2017.

15518 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide
15519 the greatest degree of imaginable utility to the application developer or user, it is arguably better
15520 than a **core** file or some other equally obscure result. (It is also possible for implementations to
15521 encode and recognize application binaries compiled in various POSIX.1-conforming
15522 environments, and modify the semantics of the underlying system to conform to the
15523 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the
15524 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`
15525 and `_SC_VERSION` functionality.

15526 Compile-Time Symbolic Constants for System-Wide Options

15527 POSIX.1-2017 includes support in certain areas for the newly adopted policy governing options
15528 and stubs.

15529 This policy provides flexibility for implementations in how they support options. It also
15530 specifies how conforming applications can adapt to different implementations that support
15531 different sets of options. It allows the following:

- 15532 1. If an implementation has no interest in supporting an option, it does not have to provide
15533 anything associated with that option beyond the announcement that it does not support
15534 it.
- 15535 2. An implementation can support a partial or incompatible version of an option (as a non-
15536 standard extension) as long as it does not claim to support the option.
- 15537 3. An application can determine whether the option is supported. A strictly conforming
15538 application must check this announcement mechanism before first using anything
15539 associated with the option.

15540 There is an important implication of this policy. POSIX.1-2017 cannot dictate the behavior of
15541 interfaces associated with an option when the implementation does not claim to support the
15542 option. In particular, it cannot require that a function associated with an unsupported option
15543 will fail if it does not perform as specified. However, this policy does not prevent a standard

15544 from requiring certain functions to always be present, but that they shall always fail on some
 15545 implementations. The `setpgid()` function in the POSIX.1-1990 standard, for example, is
 15546 considered appropriate.

15547 The POSIX standards include various options, and the C-language binding support for an
 15548 option implies that the implementation must supply data types and function interfaces. An
 15549 application must be able to discover whether the implementation supports each option.

15550 Any application must consider the following three cases for each option:

15551 1. Option never supported.

15552 The implementation advertises at compile time that the option will never be supported.
 15553 In this case, it is not necessary for the implementation to supply any of the data types or
 15554 function interfaces that are provided only as part of the option. The implementation
 15555 might provide data types and functions that are similar to those defined by POSIX.1-2017,
 15556 but there is no guarantee for any particular behavior.

15557 2. Option always supported.

15558 The implementation advertises at compile time that the option will always be supported.
 15559 In this case, all data types and function interfaces shall be available and shall operate as
 15560 specified.

15561 3. Option might or might not be supported.

15562 Some implementations might not provide a mechanism to specify support of options at
 15563 compile time. In addition, the implementation might be unable or unwilling to specify
 15564 support or non-support at compile time. In either case, any application that might use the
 15565 option at runtime must be able to compile and execute. The implementation must
 15566 provide, at compile time, all data types and function interfaces that are necessary to allow
 15567 this. In this situation, there must be a mechanism that allows the application to query, at
 15568 runtime, whether the option is supported. If the application attempts to use the option
 15569 when it is not supported, the result is unspecified unless explicitly specified otherwise in
 15570 POSIX.1-2017.

15571 **FUTURE DIRECTIONS**

15572 None.

15573 **SEE ALSO**

15574 <limits.h>, <stddef.h>, <stdint.h>, <stdio.h>, <sys/socket.h>, <sys/types.h>, <termios.h>,
 15575 <wctype.h>

15576 XSH *access()*, *alarm()*, *chown()*, *close()*, *confstr()*, *crypt()*, *ctermid()*, *dup()*, *_Exit()*, *encrypt()*, *exec*,
 15577 *fchdir()*, *fchown()*, *fdatasync()*, *fork()*, *fpathconf()*, *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*,
 15578 *geteuid()*, *getgid()*, *getgroups()*, *gethostid()*, *gethostname()*, *getlogin()*, *getopt()*, *getpgid()*, *getpgrp()*,
 15579 *getpid()*, *getppid()*, *getsid()*, *getuid()*, *isatty()*, *lchown()*, *link()*, *lockf()*, *lseek()*, *nice()*, *pause()*,
 15580 *pipe()*, *read()*, *readlink()*, *rmdir()*, *setegid()*, *seteuid()*, *setgid()*, *setpgid()*, *setpgrp()*, *setregid()*,
 15581 *setreuid()*, *setsid()*, *setuid()*, *sleep()*, *swab()*, *symlink()*, *sync()*, *sysconf()*, *tcgetpgrp()*, *tcsetpgrp()*,
 15582 *truncate()*, *ttyname()*, *unlink()*, *write()*

15583 **CHANGE HISTORY**

15584 First released in Issue 1. Derived from Issue 1 of the SVID.

15585 **Issue 5**

15586 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 15587 Threads Extension.

15588 The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added.

15589 _POSIX2_C_BIND, _XOPEN_ENH_I18N, and _XOPEN_SHM must now be set to a value other
15590 than -1 by a conforming implementation.

15591 Large File System extensions are added.

15592 The type of the argument to *sbrk()* is changed from **int** to **intptr_t**.

15593 _XBS_ constants are added to the list of constants for Options and Option Groups, to the list of
15594 constants for the *confstr()* function, and to the list of constants to the *sysconf()* function. These
15595 are all marked EX.

15596 **Issue 6**

15597 _POSIX2_C_VERSION is removed.

15598 The Open Group Corrigendum U026/4 is applied, adding the prototype for *fdatasync()*.

15599 The Open Group Corrigendum U026/1 is applied, adding the symbols _SC_XOPEN_LEGACY,
15600 _SC_XOPEN_REALTIME, and _SC_XOPEN_REALTIME_THREADS.

15601 The symbols _XOPEN_STREAMS and _SC_XOPEN_STREAMS are added to support the XSI
15602 STREAMS Option Group.

15603 Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in
15604 IEEE Std 1003.1-2001.

15605 The LEGACY symbol _SC_PASS_MAX is removed.

15606 The following new requirements on POSIX implementations derive from alignment with the
15607 Single UNIX Specification:

15608 The _CS_POSIX_* and _CS_XBS5_* constants are added for the *confstr()* function.

15609 The _SC_XBS5_* constants are added for the *sysconf()* function.

15610 The symbolic constants F_ULOCK, F_LOCK, F_TLOCK, and F_TEST are added.

15611 The **uid_t**, **gid_t**, **off_t**, **pid_t**, and **useconds_t** types are mandated.

15612 The *gethostname()* prototype is added for sockets.

15613 A new section is added for System-Wide Options.

15614 Function prototypes for *setegid()* and *seteuid()* are added.

15615 Option symbolic constants are added for _POSIX_ADVISORY_INFO, _POSIX_CPUTIME,
15616 _POSIX_SPAWN, _POSIX_SPORADIC_SERVER, _POSIX_THREAD_CPUTIME,
15617 _POSIX_THREAD_SPORADIC_SERVER, and _POSIX_TIMEOUTS, and *pathconf()* variables are
15618 added for _PC_ALLOC_SIZE_MIN, _PC_REC_INCR_XFER_SIZE, _PC_REC_MAX_XFER_SIZE,
15619 _PC_REC_MIN_XFER_SIZE, and _PC_REC_XFER_ALIGN for alignment with IEEE Std
15620 1003.1d-1999.

15621 The following are added for alignment with IEEE Std 1003.1j-2000:

15622 Option symbolic constants _POSIX_BARRIERS, _POSIX_CLOCK_SELECTION,
15623 _POSIX_MONOTONIC_CLOCK, _POSIX_READER_WRITER_LOCKS,
15624 _POSIX_SPIN_LOCKS, and _POSIX_TYPED_MEMORY_OBJECTS

15625 *sysconf()* variables _SC_BARRIERS, _SC_CLOCK_SELECTION,
15626 _SC_MONOTONIC_CLOCK, _SC_READER_WRITER_LOCKS, _SC_SPIN_LOCKS, and
15627 _SC_TYPED_MEMORY_OBJECTS

15628 The _SC_XBS5 macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,
15629 and new equivalent _SC_V6 macros associated with the ISO/IEC 9899:1999 standard are

15630 introduced.

15631 The `getwd()` function is marked LEGACY.

15632 The **restrict** keyword is added to the prototypes for `readlink()` and `swab()`.

15633 Constants for options are now harmonized, so when supported they take the year of approval of
15634 IEEE Std 1003.1-2001 as the value.

15635 The following are added for alignment with IEEE Std 1003.1q-2000:

15636 Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,
15637 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`

15638 The `sysconf()` symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,
15639 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`

15640 The `brk()` and `sbrk()` LEGACY functions are removed.

15641 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning
15642 information.

15643 The Open Group Base Resolution bwg2001-008 is applied, changing the `namelen` parameter for
15644 `gethostname()` from `socklen_t` to `size_t`.

15645 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack
15646 Address Size” to “Thread Stack Size Attribute”.

15647 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the `_POSIX_IPV6`,
15648 `_SC_V6`, and `_SC_RAW_SOCKETS` symbols.

15649 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description
15650 in “Constants for Functions” for the `_CS_POSIX_V6_LP64_OFF64_CFLAGS`,
15651 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`, and `_CS_POSIX_V6_LP64_OFF64_LIBS` symbols.

15652 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for
15653 the `_PC*` and `_SC*` constants, since these are mandatory on all implementations.

15654 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the
15655 `_PC_SYMLINK_MAX` and `_SC_SYMLINK_MAX` constants.

15656 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and
15657 margin code for the `fsync()` function.

15658 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to
15659 the APPLICATION USAGE section: “New applications should not use `_XOPEN_SHM` or
15660 `_XOPEN_ENH_I18N`.”.

15661 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements
15662 for when constants for Options and Option Groups can be defined or undefined.

15663 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the
15664 `_V6_ILP32_OFF32`, `_V6_ILP32_OFFBIG`, `_V6_LP64_OFF64`, and `_V6_LP64_OFFBIG` symbols to
15665 `_POSIX_V6_ILP32_OFF32`, `_POSIX_V6_ILP32_OFFBIG`, `_POSIX_V6_LP64_OFF64`, and
15666 `_POSIX_V6_LP64_OFFBIG`, respectively. This is for consistency with the `sysconf()` and `c99`
15667 reference pages.

15668 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of
15669 names of programming environments can be obtained using the `getconf -v` option.

15670 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the
15671 `_SC_FILE_LOCKING`, `_SC_2_C_VERSION`, and `_SC_XOPEN_XCU_VERSION` constants.

15672 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding
15673 `_SC_SS_REPL_MAX`, `_SC_TRACE_EVENT_NAME_MAX`, `_SC_TRACE_NAME_MAX`,
15674 `_SC_TRACE_SYS_MAX`, and `_SC_TRACE_USER_EVENT_MAX` to the list of symbolic constants
15675 for `sysconf()`.

15676 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for
15677 the `symlink()` function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.

15678 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding `_PC_2_SYMLINKS`
15679 to the symbolic constants list for `pathconf()`. This corresponds to the definition of
15680 `POSIX2_SYMLINKS` in the Shell and Utilities volume of IEEE Std 1003.1-2001.

15681 **Issue 7**

15682 Austin Group Interpretations 1003.1-2001 #026 and #047 are applied.

15683 Austin Group Interpretation 1003.1-2001 #166 is applied to permit an additional compiler flag to
15684 enable threads.

15685 Austin Group Interpretation 1003.1-2001 #178 is applied, clarifying the values allowed for
15686 `_POSIX2_CHAR_TERM`.

15687 SD5-XBD-ERN-41 is applied, adding the `_POSIX2_SYMLINKS` constant.

15688 SD5-XBD-ERN-76 and SD5-XBD-ERN-77 are applied.

15689 Symbols to support the UUCP Utilities option are added.

15690 The variables for the supported programming environments are updated to be V7.

15691 The `LEGACY` and obsolescent symbols are removed.

15692 The `faccessat()`, `fchownat()`, `fexecve()`, `linkat()`, `readlinkat()`, `symlinkat()`, and `unlinkat()` functions
15693 are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

15694 The `_POSIX_TRACE*` constants from the Trace option are marked obsolescent.

15695 The `_POSIX2_PBS*` constants from the Batch Environment Services and Utilities option are
15696 marked obsolescent.

15697 Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory
15698 Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks,
15699 Threads, Timeouts, and Timers options is moved to the Base.

15700 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options
15701 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex
15702 or Robust Mutex Priority Inheritance, respectively.

15703 This reference page is clarified with respect to macros and symbolic constants.

15704 Changes are made related to support for finegrained timestamps and the
15705 `_POSIX_TIMESTAMP_RESOLUTION` constant is added.

15706 The `_SC_THREAD_ROBUST_PRIO_INHERIT` and `_SC_THREAD_ROBUST_PRIO_PROTECT`
15707 symbolic constants are added.

15708 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0078 [311], XBD/TC1-2008/0079 [209],
15709 and XBD/TC1-2008/0080 [360] are applied.

15710 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0085 [783], XBD/TC2-2008/0086 [911],
15711 and XBD/TC2-2008/0087 [566] are applied.

15712 **NAME**

15713 utime.h — access and modification times structure

15714 **SYNOPSIS**15715 OB `#include <utime.h>`15716 **DESCRIPTION**15717 The <utime.h> header shall declare the **utimbuf** structure, which shall include the following
15718 members:15719 `time_t actime` Access time.
15720 `time_t modtime` Modification time.

15721 The times shall be measured in seconds since the Epoch.

15722 The <utime.h> header shall define the **time_t** type as described in <sys/types.h>.15723 The following shall be declared as a function and may also be defined as a macro. A function
15724 prototype shall be provided.15725 `int utime(const char *, const struct utimbuf *);`15726 **APPLICATION USAGE**15727 The *utime()* function only allows setting file timestamps to the nearest second. Applications
15728 should use the *utimensat()* function instead. See <sys/stat.h>.15729 **RATIONALE**

15730 None.

15731 **FUTURE DIRECTIONS**

15732 The <utime.h> header may be removed in a future version.

15733 **SEE ALSO**15734 [<sys/stat.h>](#), [<sys/types.h>](#)15735 XSH *futimens()*, *utime()*15736 **CHANGE HISTORY**

15737 First released in Issue 3.

15738 **Issue 6**15739 The following new requirements on POSIX implementations derive from alignment with the
15740 Single UNIX Specification:15741 The **time_t** type is defined.15742 **Issue 7**

15743 The <utime.h> header is marked obsolescent.

15744 **NAME**

15745 utmpx.h ‡user accounting database definitions

15746 **SYNOPSIS**

15747 XSI #include <utmpx.h>

15748 **DESCRIPTION**15749 The **<utmpx.h>** header shall define the **utmpx** structure that shall include at least the following
15750 members:

15751	char	ut_user[]	User login name.
15752	char	ut_id[]	Unspecified initialization process identifier.
15753	char	ut_line[]	Device name.
15754	pid_t	ut_pid	Process ID.
15755	short	ut_type	Type of entry.
15756	struct timeval	ut_tv	Time entry was made.

15757 The **<utmpx.h>** header shall define the **pid_t** type through **typedef**, as described in
15758 **<sys/types.h>**.15759 The **<utmpx.h>** header shall define the **timeval** structure as described in **<sys/time.h>**.15760 Inclusion of the **<utmpx.h>** header may also make visible all symbols from **<sys/time.h>**.15761 The **<utmpx.h>** header shall define the following symbolic constants as possible values for the
15762 *ut_type* member of the **utmpx** structure:

15763	EMPTY	No valid user accounting information.
15764	BOOT_TIME	Identifies time of system boot.
15765	OLD_TIME	Identifies time when system clock changed.
15766	NEW_TIME	Identifies time after system clock changed.
15767	USER_PROCESS	Identifies a process.
15768	INIT_PROCESS	Identifies a process spawned by the init process.
15769	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
15770	DEAD_PROCESS	Identifies a session leader who has exited.

15771 The following shall be declared as functions and may also be defined as macros. Function
15772 prototypes shall be provided.

```

15773 void          endutxent(void);
15774 struct utmpx *getutxent(void);
15775 struct utmpx *getutxid(const struct utmpx *);
15776 struct utmpx *getutxline(const struct utmpx *);
15777 struct utmpx *pututxline(const struct utmpx *);
15778 void          setutxent(void);

```

15779 **APPLICATION USAGE**
15780 None.

15781 **RATIONALE**
15782 None.

15783 **FUTURE DIRECTIONS**
15784 None.

15785 **SEE ALSO**
15786 [<sys/time.h>](#), [<sys/types.h>](#)
15787 XSH *endutxent()*

15788 **CHANGE HISTORY**
15789 First released in Issue 4, Version 2.

15790 **NAME**15791 `wchar.h` — wide-character handling15792 **SYNOPSIS**15793 `#include <wchar.h>`15794 **DESCRIPTION**

15795 CX Some of the functionality described on this reference page extends the ISO C standard.
 15796 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
 15797 enable the visibility of these symbols in this header.

15798 The **<wchar.h>** header shall define the following types:15799 CX **FILE** As described in **<stdio.h>**.15800 CX **locale_t** As described in **<locale.h>**.

15801 **mbstate_t** An object type other than an array type that can hold the conversion state
 15802 information necessary to convert between sequences of (possibly multi-byte)
 15803 CX characters and wide characters. If a codeset is being used such that an
 15804 **mbstate_t** needs to preserve more than two levels of reserved state, the results
 15805 are unspecified.

15806 **size_t** As described in **<stddef.h>**.15807 CX **va_list** As described in **<stdarg.h>**.15808 **wchar_t** As described in **<stddef.h>**.

15809 OB XSI **wctype_t** A scalar type of a data object that can hold values which represent locale-
 15810 specific character classification.

15811 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

15812 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are
 15813 described in the **<time.h>** header.

15814 The implementation shall support one or more programming environments in which the width
 15815 of **wint_t** is no greater than the width of type **long**. The names of these programming
 15816 environments can be obtained using the *confstr()* function or the *getconf* utility.

15817 The **<wchar.h>** header shall define the following macros:15818 **WCHAR_MAX** As described in **<stdint.h>**.15819 **WCHAR_MIN** As described in **<stdint.h>**.

15820 **WEOF** Constant expression of type **wint_t** that is returned by several WP functions to
 15821 indicate end-of-file.

15822 **NULL** As described in **<stddef.h>**.

15823 CX Inclusion of the **<wchar.h>** header may make visible all symbols from the headers **<ctype.h>**,
 15824 **<string.h>**, **<stdarg.h>**, **<stddef.h>**, **<stdio.h>**, **<stdlib.h>**, and **<time.h>**.

15825 The following shall be declared as functions and may also be defined as macros. Function
 15826 prototypes shall be provided for use with ISO C standard compilers. Arguments to functions in
 15827 this list can point to arrays containing **wchar_t** values that do not correspond to members of the
 15828 character set of the current locale. Such values shall be processed according to the specified
 15829 semantics, unless otherwise stated.

15830 `wint_t` `btowc(int);`15831 `wint_t` `fgetwc(FILE *);`

15832	wchar_t	*fgetws(wchar_t *restrict, int, FILE *restrict);
15833	wint_t	fputwc(wchar_t, FILE *);
15834	int	fputws(const wchar_t *restrict, FILE *restrict);
15835	int	fwide(FILE *, int);
15836	int	fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15837	int	fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15838	wint_t	getwc(FILE *);
15839	wint_t	getwchar(void);
15840	OB XSI	int iswalnum(wint_t);
15841	int	iswalph(wint_t);
15842	int	iswcntrl(wint_t);
15843	int	iswctype(wint_t, wctype_t);
15844	int	iswdigit(wint_t);
15845	int	iswgraph(wint_t);
15846	int	iswlower(wint_t);
15847	int	iswprint(wint_t);
15848	int	iswpunct(wint_t);
15849	int	iswspace(wint_t);
15850	int	iswupper(wint_t);
15851	int	iswxdigit(wint_t);
15852	size_t	mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15853	size_t	mbrtowc(wchar_t *restrict, const char *restrict, size_t, mbstate_t *restrict);
15854		mbstate_t *restrict);
15855	int	mbsinit(const mbstate_t *);
15856	CX	size_t mbsnrtowcs(wchar_t *restrict, const char **restrict, size_t, size_t, mbstate_t *restrict);
15857		size_t, size_t, mbstate_t *restrict);
15858	size_t	mbsrtowcs(wchar_t *restrict, const char **restrict, size_t, mbstate_t *restrict);
15859		mbstate_t *restrict);
15860	CX	FILE *open_wmemstream(wchar_t **, size_t *);
15861	wint_t	putwc(wchar_t, FILE *);
15862	wint_t	putwchar(wchar_t);
15863	int	swprintf(wchar_t *restrict, size_t, const wchar_t *restrict, ...);
15864		const wchar_t *restrict, ...);
15865	int	swscanf(const wchar_t *restrict, const wchar_t *restrict, ...);
15866		const wchar_t *restrict, ...);
15867	OB XSI	wint_t towlower(wint_t);
15868	wint_t	towupper(wint_t);
15869	wint_t	ungetwc(wint_t, FILE *);
15870	int	vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15871	int	vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15872	int	vswprintf(wchar_t *restrict, size_t, const wchar_t *restrict, va_list);
15873		const wchar_t *restrict, va_list);
15874	int	vswscanf(const wchar_t *restrict, const wchar_t *restrict, va_list);
15875		va_list);
15876	int	vwprintf(const wchar_t *restrict, va_list);
15877	int	vwscanf(const wchar_t *restrict, va_list);
15878	CX	wchar_t *wcpcpy(wchar_t *restrict, const wchar_t *restrict);
15879	wchar_t	*wcpncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15880	size_t	wcrtomb(char *restrict, wchar_t, mbstate_t *restrict);
15881	CX	int wcscasecmp(const wchar_t *, const wchar_t *);
15882	int	wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
15883	wchar_t	*wcscat(wchar_t *restrict, const wchar_t *restrict);

```
15884     wchar_t      *wcschr(const wchar_t *, wchar_t);
15885     int          wscmp(const wchar_t *, const wchar_t *);
15886     int          wscoll(const wchar_t *, const wchar_t *);
15887 CX     int          wscoll_l(const wchar_t *, const wchar_t *, locale_t);
15888     wchar_t      *wcsncpy(wchar_t *restrict, const wchar_t *restrict);
15889     size_t       wcsncpy_s(const wchar_t *, const wchar_t *);
15890 CX     wchar_t      *wcsdup(const wchar_t *);
15891     size_t       wcsftime(wchar_t *restrict, size_t,
15892                          const wchar_t *restrict, const struct tm *restrict);
15893     size_t       wcslen(const wchar_t *);
15894 CX     int          wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
15895     int          wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t,
15896                               locale_t);
15897     wchar_t      *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15898     int          wcsncmp(const wchar_t *, const wchar_t *, size_t);
15899     wchar_t      *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15900 CX     size_t       wcsnlen(const wchar_t *, size_t);
15901     size_t       wcsnrtombs(char *restrict, const wchar_t **restrict, size_t,
15902                          size_t, mbstate_t *restrict);
15903     wchar_t      *wcpbrk(const wchar_t *, const wchar_t *);
15904     wchar_t      *wcrchr(const wchar_t *, wchar_t);
15905     size_t       wcsrtoombs(char *restrict, const wchar_t **restrict,
15906                          size_t, mbstate_t *restrict);
15907     size_t       wcsspn(const wchar_t *, const wchar_t *);
15908     wchar_t      *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15909     double       wcstod(const wchar_t *restrict, wchar_t **restrict);
15910     float        wcstof(const wchar_t *restrict, wchar_t **restrict);
15911     wchar_t      *wcstok(wchar_t *restrict, const wchar_t *restrict,
15912                          wchar_t **restrict);
15913     long         wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15914     long double  wcstold(const wchar_t *restrict, wchar_t **restrict);
15915     long long    wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
15916     unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15917     unsigned long long
15918     wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15919 XSI     int          wcswidth(const wchar_t *, size_t);
15920     size_t       wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15921 CX     size_t       wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
15922                              size_t, locale_t);
15923     int          wctob(wint_t);
15924 OB XSI  wctype_t     wctype(const char *);
15925 XSI     int          wcwidth(wchar_t);
15926     wchar_t      *wmemchr(const wchar_t *, wchar_t, size_t);
15927     int          wmemcmp(const wchar_t *, const wchar_t *, size_t);
15928     wchar_t      *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15929     wchar_t      *wmemmove(wchar_t *, const wchar_t *, size_t);
15930     wchar_t      *wmemset(wchar_t *, wchar_t, size_t);
15931     int          wprintf(const wchar_t *restrict, ...);
15932     int          wscanf(const wchar_t *restrict, ...);
```

15933 **APPLICATION USAGE**

15934 The *iswblank()* function was a late addition to the ISO C standard and was introduced at the
 15935 same time as the ISO C standard introduced <wctype.h>, which contains all of the *isw*()*
 15936 functions. The Open Group Base Specifications had previously aligned with the MSE working
 15937 draft and had introduced the rest of the *isw*()* functions into <wchar.h>. For backwards-
 15938 compatibility, the original set of *isw*()* functions, without *iswblank()*, are permitted (as part of
 15939 the XSI option) in <wchar.h>. For maximum portability, applications should include
 15940 <wctype.h> in order to obtain declarations for the *isw*()* functions. This compatibility has been
 15941 made obsolescent.

15942 **RATIONALE**

15943 In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their
 15944 presence here is thus an extension.

15945 **FUTURE DIRECTIONS**

15946 None.

15947 **SEE ALSO**

15948 <ctype.h>, <locale.h>, <stdarg.h>, <stddef.h>, <stdint.h>, <stdio.h>, <stdlib.h>, <string.h>,
 15949 <time.h>, <wctype.h>

15950 XSH Section 2.2 (on page 472), *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwide()*,
 15951 *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpunct()*, *iswctype()*, *iswdigit()*,
 15952 *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *mbrlen()*,
 15953 *mbrtowc()*, *mbsinit()*, *mbsrtowcs()*, *open_memstream()*, *putwc()*, *putwchar()*, *towlower()*,
 15954 *towupper()*, *ungetwc()*, *vwprintf()*, *vwscanf()*, *wcrtomb()*, *wcscasecmp()*, *wcscat()*, *wcschr()*,
 15955 *wcscmp()*, *wcscoll()*, *wcscpy()*, *wcscspn()*, *wcsdup()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*,
 15956 *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsrtombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstok()*, *wcstol()*,
 15957 *wcstoul()*, *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*,
 15958 *wmemcpy()*, *wmemmove()*, *wmemset()*

15959 XCU *getconf*

15960 **CHANGE HISTORY**

15961 First released in Issue 4.

15962 **Issue 5**

15963 Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15964 **Issue 6**

15965 The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and
 15966 *wcwidth()* are marked as extensions.

15967 The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()*
 15968 function.

15969 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

15970 Various function prototypes are updated to add the **restrict** keyword.

15971 The functions *vwscanf()*, *vsscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are
 15972 added.

15973 The type **wctype_t**, the *isw*()*, *to*()*, and *wctype()* functions are marked as XSI extensions.

15974 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION
 15975 USAGE section.

15976 **Issue 7**

15977 The *mbsnrtowcs()*, *open_wmemstream()*, *wcpcpy()*, *wcpncpy()*, *wscasecmp()*, *wcsdup()*,
15978 *wscncasecmp()*, *wcsnlen()*, and *wcsnrtombs()* functions are added from The Open Group
15979 Technical Standard, 2006, Extended API Set Part 1.

15980 The *wscasecmp_l()*, *wscncasecmp_l()*, *wscoll_l()*, and *wcsxfrm_l()* functions are added from The
15981 Open Group Technical Standard, 2006, Extended API Set Part 4.

15982 The **wctype_t** type, and the *isw*()*, *towlower()*, and *towupper()* functions are marked obsolescent
15983 in **<wchar.h>** since the ISO C standard requires the declarations to be in **<wctype.h>**.

15984 This reference page is clarified with respect to macros and symbolic constants, and a declaration
15985 for the **locale_t** type is added.

15986 POSIX.1-2008, Technical Corrigendum 1, XBD/TC1-2008/0081 [380] is applied.

15987 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0088 [73] is applied.

15988 **NAME**

15989 wctype.h ‡wide-character classification and mapping utilities

15990 **SYNOPSIS**

15991 #include <wctype.h>

15992 **DESCRIPTION**

15993 CX Some of the functionality described on this reference page extends the ISO C standard.
15994 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 472) to
15995 enable the visibility of these symbols in this header.

15996 The <wctype.h> header shall define the following types:

15997 **wint_t** As described in <wchar.h>.

15998 **wctrans_t** A scalar type that can hold values which represent locale-specific character
15999 mappings.

16000 **wctype_t** As described in <wchar.h>.

16001 CX The <wctype.h> header shall define the **locale_t** type as described in <locale.h>.

16002 The <wctype.h> header shall define the following macro:

16003 **WEOF** As described in <wchar.h>.

16004 For all functions described in this header that accept an argument of type **wint_t**, the value is
16005 representable as a **wchar_t** or equals the value of WEOF. If this argument has any other value,
16006 the behavior is undefined.

16007 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

16008 CX Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>,
16009 <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>.

16010 The following shall be declared as functions and may also be defined as macros. Function
16011 prototypes shall be provided for use with ISO C standard compilers.

```

16012       int           iswalnum(wint_t);
16013 CX       int           iswalnum_l(wint_t, locale_t);
16014       int           iswalpha(wint_t);
16015 CX       int           iswalpha_l(wint_t, locale_t);
16016       int           iswblank(wint_t);
16017 CX       int           iswblank_l(wint_t, locale_t);
16018       int           iswcntrl(wint_t);
16019 CX       int           iswcntrl_l(wint_t, locale_t);
16020       int           iswctype(wint_t, wctype_t);
16021 CX       int           iswctype_l(wint_t, wctype_t, locale_t);
16022       int           iswdigit(wint_t);
16023 CX       int           iswdigit_l(wint_t, locale_t);
16024       int           iswgraph(wint_t);
16025 CX       int           iswgraph_l(wint_t, locale_t);
16026       int           iswlower(wint_t);
16027 CX       int           iswlower_l(wint_t, locale_t);
16028       int           iswprint(wint_t);
16029 CX       int           iswprint_l(wint_t, locale_t);
16030       int           iswpunct(wint_t);
16031 CX       int           iswpunct_l(wint_t, locale_t);
16032       int           iswspace(wint_t);
    
```

```

16033 CX      int      iswspace_l(wint_t, locale_t);
16034         int      iswupper(wint_t);
16035 CX      int      iswupper_l(wint_t, locale_t);
16036         int      iswxdigit(wint_t);
16037 CX      int      iswxdigit_l(wint_t, locale_t);
16038         wint_t    towctrans(wint_t, wctrans_t);
16039 CX      wint_t    towctrans_l(wint_t, wctrans_t, locale_t);
16040         wint_t    towlower(wint_t);
16041 CX      wint_t    towlower_l(wint_t, locale_t);
16042         wint_t    towupper(wint_t);
16043 CX      wint_t    towupper_l(wint_t, locale_t);
16044         wctrans_t wctrans(const char *);
16045 CX      wctrans_t wctrans_l(const char *, locale_t);
16046         wctype_t  wctype(const char *);
16047 CX      wctype_t  wctype_l(const char *, locale_t);

```

16048 APPLICATION USAGE

16049 None.

16050 RATIONALE

16051 None.

16052 FUTURE DIRECTIONS

16053 None.

16054 SEE ALSO

16055 [<ctype.h>](#), [<locale.h>](#), [<stdarg.h>](#), [<stddef.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<string.h>](#), [<time.h>](#),
16056 [<wchar.h>](#)

16057 XSH Section 2.2 (on page 472), [iswalnum\(\)](#), [iswalpha\(\)](#), [iswblank\(\)](#), [iswcntrl\(\)](#), [iswctype\(\)](#),
16058 [iswdigit\(\)](#), [iswgraph\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#), [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#),
16059 [setlocale\(\)](#), [towctrans\(\)](#), [towlower\(\)](#), [towupper\(\)](#), [wctrans\(\)](#), [wctype\(\)](#)

16060 CHANGE HISTORY

16061 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

16062 Issue 6

16063 The [iswblank\(\)](#) function is added for alignment with the ISO/IEC 9899:1999 standard.

16064 Issue 7

16065 SD5-XBD-ERN-6 is applied.

16066 The [*_l\(\)](#) functions are added from The Open Group Technical Standard, 2006, Extended API Set
16067 Part 4.

16068 This reference page is clarified with respect to macros and symbolic constants.

16069 **NAME**

16070 wordexp.h — word-expansion types

16071 **SYNOPSIS**

16072 #include <wordexp.h>

16073 **DESCRIPTION**

16074 The <wordexp.h> header shall define the structures and symbolic constants used by the
16075 *wordexp()* and *wordfree()* functions.

16076 The <wordexp.h> header shall define the **wordexp_t** structure type, which shall include at least
16077 the following members:

16078 size_t we_wordc Count of words matched by *words*.
16079 char **we_wordv Pointer to list of expanded words.
16080 size_t we_offs Slots to reserve at the beginning of *we_wordv*.

16081 The <wordexp.h> header shall define the following symbolic constants for use as flags for the
16082 *wordexp()* function:

- 16083 WRDE_APPEND Append words to those previously generated.
- 16084 WRDE_DOOFFS Number of null pointers to prepend to *we_wordv*.
- 16085 WRDE_NOCMD Fail if command substitution is requested.
- 16086 WRDE_REUSE The *pwordexp* argument was passed to a previous successful call to
16087 *wordexp()*, and has not been passed to *wordfree()*. The result is the same
16088 as if the application had called *wordfree()* and then called *wordexp()*
16089 without WRDE_REUSE.
- 16090 WRDE_SHOWERR Do not redirect *stderr* to **/dev/null**.
- 16091 WRDE_UNDEF Report error on an attempt to expand an undefined shell variable.

16092 The <wordexp.h> header shall define the following symbolic constants as error return values:

- 16093 WRDE_BADCHAR One of the unquoted characters ‡<newline>,'|', '&', ';', '<', '>',
16094 '(', ')', '{', '}' ‡appears in *words* in an inappropriate context.
- 16095 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.
- 16096 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.
- 16097 WRDE_NOSPACE Attempt to allocate memory failed.
- 16098 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
16099 string.

16100 The <wordexp.h> header shall define the **size_t** type as described in <stddef.h>.

16101 The following shall be declared as functions and may also be defined as macros. Function
16102 prototypes shall be provided.

16103 int wordexp(const char *restrict, wordexp_t *restrict, int);
16104 void wordfree(wordexp_t *);

16105 APPLICATION USAGE

16106 None.

16107 RATIONALE

16108 None.

16109 FUTURE DIRECTIONS

16110 None.

16111 SEE ALSO

16112 [<stddef.h>](#)

16113 XSH [Section 2.6](#)

16114 CHANGE HISTORY

16115 First released in Issue 4. Derived from the ISO POSIX-2 standard.

16116 Issue 6

16117 The **restrict** keyword is added to the prototype for *wordexp()*.

16118 The WRDE_NOSYS constant is marked obsolescent.

16119 Issue 7

16120 The obsolescent WRDE_NOSYS constant is removed.

16121 This reference page is clarified with respect to macros and symbolic constants.

16122

 *Open Group Standard*

16123

Vol. 2:

16124

System Interfaces, Issue 7

16125

The Open Group

16126

The Institute of Electrical and Electronics Engineers, Inc.

16127

16128

Introduction

16129

The System Interfaces volume of POSIX.1-2017 describes the interfaces offered to application programs by POSIX-conformant systems.

16130

1.1 Relationship to Other Formal Standards

16132

Great care has been taken to ensure that this volume of POSIX.1-2017 is fully aligned with the following standards:

16133

16134

ISO C (1999)

16135

ISO/IEC 9899:1999, Programming Language C, including ISO/IEC

16136

9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC

16137

9899:1999/Cor.3.

16138

Parts of the ISO/IEC 9899:1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also mandated by this volume of POSIX.1-2017. Some functions and headers included within this volume of POSIX.1-2017 have a version in the ISO C standard; in this case CX markings are added as appropriate to show where the ISO C standard has been extended (see [Section 1.7.1](#), on page 7). Any conflict between this volume of POSIX.1-2017 and the ISO C standard is unintentional.

16139

16140

16141

16142

16143

16144

This volume of POSIX.1-2017 also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

16145

1.2 Format of Entries

16147

The entries in [Chapter 3](#) are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

16148

16149

NAME

16150

This section gives the name or names of the entry and briefly states its purpose.

16151

SYNOPSIS

16152

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names of such headers are shown, for example:

16153

16154

16155

```
#include <stdio.h>
```

16156

DESCRIPTION

16157

This section describes the functionality of the function or header.

16158

RETURN VALUE

16159

This section indicates the possible return values, if any.

16160

If the implementation can detect errors, “successful completion” means that no error

16161 has been detected during execution of the function. If the implementation does detect
16162 an error, the error is indicated.

16163 For functions where no errors are defined, “successful completion” means that if the
16164 implementation checks for errors, no error has been detected. If the implementation can
16165 detect errors, and an error is detected, the indicated return value is returned and *errno*
16166 may be set.

16167 **ERRORS**

16168 This section gives the symbolic names of the error values returned by a function or
16169 stored into a variable accessed through the symbol *errno* if an error occurs.

16170 “No errors are defined” means that error values returned by a function or stored into a
16171 variable accessed through the symbol *errno*, if any, depend on the implementation.

16172 **EXAMPLES**

16173 This section is informative.

16174 This section gives examples of usage, where appropriate. In the event of conflict
16175 between an example and a normative part of this volume of POSIX.1-2017, the
16176 normative material is to be taken as correct.

16177 **APPLICATION USAGE**

16178 This section is informative.

16179 This section gives warnings and advice to application developers about the entry. In the
16180 event of conflict between warnings and advice and a normative part of this volume of
16181 POSIX.1-2017, the normative material is to be taken as correct.

16182 **RATIONALE**

16183 This section is informative.

16184 This section contains historical information concerning the contents of this volume of
16185 POSIX.1-2017 and why features were included or discarded by the standard
16186 developers.

16187 **FUTURE DIRECTIONS**

16188 This section is informative.

16189 This section provides comments which should be used as a guide to current thinking;
16190 there is not necessarily a commitment to adopt these future directions.

16191 **SEE ALSO**

16192 This section is informative.

16193 This section gives references to related information.

16194 **CHANGE HISTORY**

16195 This section is informative.

16196 This section shows the derivation of the entry and any significant changes that have
16197 been made to it.

16198

Chapter 2

16199

General Information

16200

This chapter covers information that is relevant to all the functions specified in [Chapter 3](#) and [XBD Chapter 13](#) (on page 219).

16201

16202 2.1 Use and Implementation of Interfaces

16203 2.1.1 Use and Implementation of Functions

16204

Each of the following statements shall apply to all functions unless explicitly stated otherwise in the detailed descriptions that follow:

16205

16206

1. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behavior is undefined.

16207

16208

16209

2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the <left-parenthesis> that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language **#undef** construct to remove any such macro definition shall also ensure that an actual function is referred to.

16210

16211

16212

16213

16214

16215

16216

16217

3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments.

16218

16219

16220

4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.

16221

16222

16223

5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

16224

16225 2.1.2 Use and Implementation of Macros

16226 Each of the following statements shall apply to all macros unless explicitly stated otherwise:

- 16227 1. Any definition of an object-like macro in a header shall expand to code that is fully
16228 protected by parentheses where necessary, so that it groups in an arbitrary expression as
16229 if it were a single identifier.
- 16230 2. All object-like macros listed as expanding to integer constant expressions shall
16231 additionally be suitable for use in **#if** preprocessing directives.
- 16232 3. Any definition of a function-like macro in a header shall expand to code that evaluates
16233 each of its arguments exactly once, fully protected by parentheses where necessary, so
16234 that it is generally safe to use arbitrary expressions as arguments.
- 16235 4. Any definition of a function-like macro in a header can be invoked in an expression
16236 anywhere a function with a compatible return type could be called.

16237 2.2 The Compilation Environment

16238 2.2.1 POSIX.1 Symbols

16239 Certain symbols in this volume of POSIX.1-2017 are defined in headers (see XBD [Chapter 13](#), on
16240 page 219). Some of those headers could also define symbols other than those defined by
16241 POSIX.1-2017, potentially conflicting with symbols used by the application. Also, POSIX.1-2017
16242 defines symbols that are not permitted by other standards to appear in those headers without
16243 some control on the visibility of those symbols.

16244 Symbols called “feature test macros” are used to control the visibility of symbols that might be
16245 included in a header. Implementations, future versions of this standard, and other standards
16246 may define additional feature test macros.

16247 In the compilation of an application that **#defines** a feature test macro specified by
16248 POSIX.1-2017, no header defined by POSIX.1-2017 shall be included prior to the definition of the
16249 feature test macro. This restriction also applies to any implementation-provided header in
16250 which these feature test macros are used. If the definition of the macro does not precede the
16251 **#include**, the result is undefined.

16252 Feature test macros shall begin with the <underscore> character ('_').

16253 2.2.1.1 *The _POSIX_C_SOURCE Feature Test Macro*

16254 A POSIX-conforming application shall ensure that the feature test macro `_POSIX_C_SOURCE` is
16255 defined before inclusion of any header.

16256 When an application includes a header described by POSIX.1-2017, and when this feature test
16257 macro is defined to have the value 200809L:

- 16258 1. All symbols required by POSIX.1-2017 to appear when the header is included shall be
16259 made visible.

16260 2. Symbols that are explicitly permitted, but not required, by POSIX.1-2017 to appear in that
16261 header (including those in reserved name spaces) may be made visible.

16262 3. Additional symbols not required or explicitly permitted by POSIX.1-2017 to be in that
16263 header shall not be made visible, except when enabled by another feature test macro.

16264 Identifiers in POSIX.1-2017 may only be undefined using the **#undef** directive as described in
16265 [Section 2.1](#) (on page 471) or [Section 2.2.2](#). These **#undef** directives shall follow all **#include**
16266 directives of any header in POSIX.1-2017.

16267 **Note:** The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been
16268 superseded by `_POSIX_C_SOURCE`.

16269 2.2.1.2 *The `_XOPEN_SOURCE` Feature Test Macro*

16270 XSI An XSI-conforming application shall ensure that the feature test macro `_XOPEN_SOURCE` is
16271 defined with the value 700 before inclusion of any header. This is needed to enable the
16272 functionality described in [Section 2.2.1.1](#) (on page 472) and to ensure that the XSI option is
16273 enabled.

16274 Since this volume of POSIX.1-2017 is aligned with the ISO C standard, and since all functionality
16275 enabled by `_POSIX_C_SOURCE` set equal to 200809L is enabled by `_XOPEN_SOURCE` set equal
16276 to 700, there should be no need to define `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so
16277 defined. Therefore, if `_XOPEN_SOURCE` is set equal to 700 and `_POSIX_C_SOURCE` is set equal
16278 to 200809L, the behavior is the same as if only `_XOPEN_SOURCE` is defined and set equal to
16279 700. However, should `_POSIX_C_SOURCE` be set to a value greater than 200809L, the behavior
16280 is unspecified.

16281 If `_XOPEN_SOURCE` is defined with the value 700 and `_POSIX_C_SOURCE` is undefined before
16282 inclusion of any header, then the header may define the `_POSIX_C_SOURCE` macro with the
16283 value 200809L.

16284 2.2.2 The Name Space

16285 All identifiers in this volume of POSIX.1-2017, except *environ*, are defined in at least one of the
16286 XSI headers, as shown in XBD [Chapter 13](#) (on page 219). When `_XOPEN_SOURCE` or
16287 `_POSIX_C_SOURCE` is defined, each header defines or declares some identifiers, potentially
16288 conflicting with identifiers used by the application. The set of identifiers visible to the
16289 application consists of precisely those identifiers from the header pages of the included headers,
16290 as well as additional identifiers reserved for the implementation. In addition, some headers may
16291 make visible identifiers from other headers as indicated on the relevant header pages.

16292 Implementations may also add members to a structure or union without controlling the
16293 visibility of those members with a feature test macro, as long as a user-defined macro with the
16294 same name cannot interfere with the correct interpretation of the program. The identifiers
16295 reserved for use by the implementation are described below:

- 16296 1. Each identifier with external linkage described in the header section is reserved for use as
16297 an identifier with external linkage if the header is included.
- 16298 2. Each macro described in the header section is reserved for any use if the header is
16299 included.
- 16300 3. Each identifier with file scope described in the header section is reserved for use as an
16301 identifier with file scope in the same name space if the header is included.

16302 The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by POSIX.1-2017 and other

16303 POSIX standards. Implementations may add symbols to the headers shown in the following
16304 table, provided the identifiers for those symbols either:

- 16305 1. Begin with the corresponding reserved prefixes in the table, or
- 16306 2. Have one of the corresponding complete names in the table, or
- 16307 3. End in the string indicated as a reserved suffix in the table and do not use the reserved
16308 prefixes `posix_`, `POSIX_`, or `_POSIX_`, as long as the reserved suffix is in that part of the
16309 name considered significant by the implementation.

16310 Symbols that use the reserved prefix `_POSIX_` may be made visible by implementations in any
16311 header defined by POSIX.1-2017.

	Header	Prefix	Suffix	Complete Name
16312				
16313				
16314	<aio.h>	aio_, lio_, AIO_, LIO_		
16315	<arpa/inet.h>	inet_		
16316	<ctype.h>	to[a-z], is[a-z]		
16317	<dlfcn.h>	RTLD_		
16318	<dirent.h>	d_		
16319	<fcntl.h>	l_		
16320	XSI <fmtmsg.h>	MM_		
16321	<fnmatch.h>	FNM_		
16322	XSI <ftw.h>	FTW		
16323	<glob.h>	gl_, GLOB_		
16324	<grp.h>	gr_		
16325	<limits.h>		_MAX, _MIN	
16326	XSI <math.h>	M_		
16327	MSG <mqueue.h>	mq_, MQ_		
16328	XSI <ndbm.h>	dbm_, DBM_		
16329	<netdb.h>	ai_, h_, n_, p_, s_		
16330	<net/if.h>	if_, IF_		
16331	<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
16332	IP6	in6_, s6_, sin6_, IPV6_		
16333	<netinet/tcp.h>	TCP_		
16334	<nl_types.h>	NL_		
16335	<poll.h>	pd_, ph_, ps_, POLL		
16336	<pthread.h>	pthread_, PTHREAD_		
16337	<pwd.h>	pw_		
16338	<regex.h>	re_, rm_, REG_		
16339	<sched.h>	sched_, SCHED_		
16340	<semaphore.h>	sem_, SEM_		
16341	CX <signal.h>	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_, ss_, sv_, SS_, TRAP_, POLL_		
16342				
16343	XSI			
16344	OB XSR			
16345	<stropts.h>	bi_, ic_, l_, sl_, str_, FLUSH[A-Z], I_, S_, SND[A-Z]		
16346				
16347	<stdlib.h>	str[a-z]		
16348	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
16349	XSI <sys/ipc.h>	ipc_, IPC_		key, pad, seq
16350	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
16351				
16352	XSI <sys/msg.h>	msg, MSG_[A-Z]		msg
16353	XSI <sys/resource.h>	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
16354	<sys/select.h>	fd_, fds_, FD_		

	Header	Prefix	Suffix	Complete Name
16355				
16356				
16357	XSI	<sys/sem.h>	sem, SEM_	sem
16358	XSI	<sys/shm.h>	shm, SHM[A-Z], SHM_[A-Z]	
16359		<sys/socket.h>	cmsg_, if_, ifc_, ifra_, ifru_, infu_, l_, msg_, sa_, ss_, AF_, MSG_, PF_, SCM_, SHUT_, SO	
16360				
16361	XSI			
16362				
16363		<sys/stat.h>	st_	
16364		<sys/statvfs.h>	f_, ST_	
16365	XSI	<sys/time.h>	it_, tv_, ITIMER_	
16366		<sys/times.h>	tms_	
16367	XSI	<sys/uio.h>	iov_	UIO_MAXIOV
16368		<sys/un.h>	sun_	
16369		<sys/utsname.h>	uts_	
16370		<sys/wait.h>	P_, W[A-Z]	
16371	XSI	<syslog.h>	LOG_	
16372		<termios.h>	c_, B[0-9], TC	
16373	CX	<time.h>	tm_ clock_, it_, timer_, tv_, CLOCK_, TIMER_	
16374				
16375				
16376	XSI	<ulimit.h>	UL_	
16377	OB	<utime.h>	utim_	
16378	XSI	<utmpx.h>	ut_	_LVL, _PROCESS, _TIME
16379				
16380		<wchar.h>	wcs[a-z]	
16381		<wctype.h>	is[a-z], to[a-z]	
16382		<wordexp.h>	we_, WRDE_	
16383	CX	ANY header		_t

16384 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the
 16385 portable character set. The notation [a-z] indicates any lowercase letter in the portable character
 16386 set. Commas and spaces in the lists of prefixes and complete names in the above table are not
 16387 part of any prefix or complete name. The ISO C standard reserves int[0-9a-z]*_t and uint[0-9a-
 16388 z]*_t in <stdint.h>; this is not included in the table above because it is covered by the reserved
 16389 _t suffix for any header.

16390 Implementations may also add symbols to the <complex.h> header with the following complete
 16391 names or the same names suffixed with 'f' or 'l':

16392	cerf	cerfc	cexp2
16393	cexpm1	clog10	clog1p
16394	clog2	clgamma	ctgamma

16395 If any header in the following table is included, macros with the prefixes shown may be defined.
 16396 After the last inclusion of a given header, an application may use identifiers with the
 16397 corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the
 16398 corresponding macro.

Header	Prefix
16399 <errno.h>	E[0-9], E[A-Z]
16400 <fcntl.h>	F_, O_
16401 <fenv.h>	FE_[A-Z]
16402 <inttypes.h>	PRI[Xa-z], SCN[Xa-z]
16403 <locale.h>	LC_[A-Z]
16404 <math.h>	FP_[A-Z]
16405 <netinet/in.h>	IMPLINK_, IN_, IP_, IPPORT_, SOCK_
16406	IN6_
16407 IP6	
16408 <signal.h>	SIG_, SIG[A-Z],
16409 XSI	SV_
16410 CX	<stdio.h> SEEK_
16411 OB XSR	<stropts.h> M_, MUXID_R[A-Z], STR
16412 XSI	<sys/resource.h> RLIM_
16413 XSI	<sys/socket.h> CMSG_
16414	<sys/stat.h> S_
16415 XSI	<sys/uio.h> IOV_
16416	<termios.h> I, O, V (See below.)
16417	<unistd.h> SEEK_

16418 The following are used to reserve complete names for the <stdint.h> header:

- 16419 INT[0-9A-Za-z-]*_MIN
- 16420 INT[0-9A-Za-z-]*_MAX
- 16421 INT[0-9A-Za-z-]*_C
- 16422 UINT[0-9A-Za-z-]*_MIN
- 16423 UINT[0-9A-Za-z-]*_MAX
- 16424 UINT[0-9A-Za-z-]*_C

16425 **Note:** The notation [0-9] indicates any digit. The notation [A-Z] indicates any uppercase letter in the
 16426 portable character set. The notation [Xa-z] indicates the character 'x' or any lowercase letter
 16427 in the portable character set. The notation [0-9A-Za-z-]* indicates zero or more occurrences of
 16428 any of the following: a digit, an uppercase or lowercase letter in the portable character set, or an
 16429 <underscore>.

16430 XSI The following reserved names are used as exact matches for <termios.h>:

16431 CBAUD	EXTB	VDSUSP
16432 DEFCH0	FLUSHO	VLNEXT
16433 ECHOCTL	LOBLK	VREPRINT
16434 ECH0KE	PENDIN	VSTATUS
16435 ECHOPRT	SWTCH	VWERASE
16436 EXTA	VDISCARD	

- 16437 The following identifiers are reserved regardless of the inclusion of headers:
- 16438 1. With the exception of identifiers beginning with the prefix `_POSIX_`, all identifiers that
16439 begin with an `<underscore>` and either an uppercase letter or another `<underscore>` are
16440 always reserved for any use by the implementation.
- 16441 2. All identifiers that begin with an `<underscore>` are always reserved for use as identifiers
16442 with file scope in both the ordinary identifier and tag name spaces.
- 16443 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
16444 Some of these identifiers do not appear in this volume of POSIX.1-2017, but are reserved for
16445 future use by the ISO C standard.
- 16446 4. All functions and external identifiers defined in XBD [Chapter 13](#) (on page 219) are reserved
16447 for use as identifiers with external linkage.
- 16448 5. All the identifiers defined in this volume of POSIX.1-2017 that have external linkage are
16449 always reserved for use as identifiers with external linkage.
- 16450 **Note:** The notation `[a-z]` indicates any lowercase letter in the portable character set. The notation `'*'`
16451 indicates any combination of digits, letters in the portable character set, or `<underscore>`.
- 16452 No other identifiers are reserved.

16453	_Exit	catan	clogf	exit	fopen
16454	abort	catanf	clogl	exp	fprintf
16455	abs	catanh	conj	exp2	fputc
16456	acos	catanhf	conjf	exp2f	fputs
16457	acosh	catanhl	conjl	exp2l	fputwc
16458	acoshf	catanl	copysign	expf	fputws
16459	acoshf	cbrt	copysignf	expl	fread
16460	acoshl	cbrtf	copysignl	expm1	free
16461	acosl	cbrtl	cos	expm1f	freopen
16462	asctime	ccos	cosf	expm1l	frexp
16463	asin	ccosf	cosh	fabs	frexpf
16464	asinf	ccosh	coshf	fabsf	frexpl
16465	asinh	ccoshf	coshl	fabsl	fscanf
16466	asinhf	ccoshl	cosl	fclose	fseek
16467	asinhl	ccosl	cpow	fdim	fsetpos
16468	asinl	ceil	cpowf	fdimf	ftell
16469	atan	ceilf	cpowl	fdiml	fwide
16470	atan2	ceil	cproj	feclearexcept	fwprintf
16471	atan2f	cerf	cprojf	fegetenv	fwrite
16472	atan2l	cerfc	cprojl	fegetexceptflag	fwscanf
16473	atanf	cerfcf	creal	fegetround	getc
16474	atanh	cerfcl	crealf	feholdexcept	getchar
16475	atanhf	cerff	creall	feof	getenv
16476	atanhl	cerfl	csin	feraiseexcept	gets
16477	atanl	cexpm1	csinf	ferror	getwc
16478	atexit	cexpm1f	csinh	fesetenv	getwchar
16479	atof	cexpm1l	csinhf	fesetexceptflag	gmtime
16480	atoi	cexp	csinhl	fesetround	hypot
16481	atol	cexp2	csinl	fetestexcept	hypotf
16482	atoll	cexp2f	csqrt	feupdateenv	hypotl
16483	bsearch	cexp2l	csqrtf	fflush	ilogb
16484	btowc	cexpf	csqrtl	fgetc	ilogbf
16485	cabs	cexpl	ctan	fgetpos	ilogbl
16486	cabsf	cimag	ctanf	fgets	imaxabs
16487	cabsl	cimagf	ctanh	fgetwc	imaxdiv
16488	cacos	cimagl	ctanhf	fgetws	is[a-z]*
16489	cacosf	clearerr	ctanhl	floor	labs
16490	cacosh	clgamma	ctanl	floorf	ldexp
16491	cacoshf	clgammaf	ctgamma	floorl	ldexpf
16492	cacoshl	clgammal	ctgammaf	fma	ldexpl
16493	cacosl	clock	ctgammal	fmaf	ldiv
16494	calloc	clog	ctime	fmal	lgamma
16495	carg	clog10	difftime	fmax	lgammaf
16496	cargf	clog10f	div	fmaxf	lgammal
16497	cargl	clog10l	erf	fmaxl	llabs
16498	casin	clog1p	erfc	fmin	lldiv
16499	casinf	clog1pf	erfcf	fminf	llrint
16500	casinh	clog1pl	erfcl	fminl	llrintf
16501	casinhf	clog2	erff	fmod	llrintl
16502	casinhl	clog2f	erfl	fmodf	llround
16503	casinl	clog2l	errno	fmodl	llroundf

16504	llroundl	mbtowc	remainderf	sprintf	vwscanf
16505	localeconv	mem[a-z]*	remainderl	sqrt	vprintf
16506	localtime	mktime	remove	sqrtf	vscanf
16507	log	modf	remquo	sqrtl	vsnprintf
16508	log10	modff	remquof	srand	vsprintf
16509	log10f	modfl	remquol	sscanf	vsscanf
16510	log10l	nan	rename	str[a-z]*	vswprintf
16511	log1p	nanf	rewind	swprintf	vswscanf
16512	log1pf	nanl	rint	swscanf	vwprintf
16513	log1pl	nearbyint	rintf	system	vwscanf
16514	log2	nearbyintf	rintl	tan	wcrtomb
16515	log2f	nearbyintl	round	tanf	wcs[a-z]*
16516	log2l	nextafter	roundf	tanh	wctob
16517	logb	nextafterf	roundl	tanhf	wctomb
16518	logbf	nextafterl	scalbln	tanhf	wctrans
16519	logbl	nexttoward	scalblnf	tanl	wctype
16520	logf	nexttowardf	scalblnl	tgamma	wmemchr
16521	logl	nexttowardl	scalbn	tgammaf	wmemcmp
16522	longjmp	perror	scalbnf	tgammaf	wmemcpy
16523	lrint	pow	scalbnl	time	wmemmove
16524	lrintf	powf	scanf	tmpfile	wmemset
16525	lrintl	powl	setbuf	tmpnam	wprintf
16526	lround	printf	setjmp	to[a-z]*	wscanf
16527	lroundf	putc	setlocale	trunc	
16528	lroundl	putchar	setvbuf	truncf	
16529	malloc	puts	signal	truncl	
16530	math_errhandling	putwc	sin	ungetc	
16531	mblen	putwchar	sinf	ungetwc	
16532	mbrlen	qsort	sinh	va_copy	
16533	mbrtowc	raise	sinhf	va_end	
16534	mbsinit	rand	sinhl	vfprintf	
16535	mbsrtowcs	realloc	sinl	vfscanf	
16536	mbstowcs	remainder	snprintf	vwprintf	

16537 **Note:** The notation [a-z] indicates any lowercase letter in the portable character set. The notation '*'
 16538 indicates any sequence of zero or more characters that are valid in identifiers with external
 16539 linkage.

16540 Applications shall not declare or define identifiers with the same name as an identifier reserved
 16541 in the same context. Since macro names are replaced whenever found, independent of scope and
 16542 name space, macro names matching any of the reserved identifier names shall not be defined by
 16543 an application if any associated header is included.

16544 Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,
 16545 headers may be included in any order, and each may be included more than once in a given
 16546 scope, with no difference in effect from that of being included only once.

16547 If used, the application shall ensure that a header is included outside of any external declaration
 16548 or definition, and it shall be first included before the first reference to any type or macro it
 16549 defines, or to any function or object it declares. However, if an identifier is declared or defined in
 16550 more than one header, the second and subsequent associated headers may be included after the
 16551 initial reference to the identifier. Prior to the inclusion of a header, the application shall not
 16552 define any macros with names lexically identical to symbols defined by that header.

16553 2.3 Error Numbers

16554 Most functions can provide an error number. The means by which each function provides its
16555 error numbers is specified in its description.

16556 Some functions provide the error number in a variable accessed through the symbol *errno*,
16557 defined by including the `<errno.h>` header. The value of *errno* should only be examined when it
16558 is indicated to be valid by a function's return value. No function in this volume of POSIX.1-2017
16559 shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by
16560 function calls or assignments to *errno* by other threads.

16561 Some functions return an error number directly as the function value. These functions return a
16562 value of zero to indicate success.

16563 If more than one error occurs in processing a function call, any one of the possible errors may be
16564 returned, as the order of detection is undefined.

16565 Implementations may support additional errors not included in this list, may generate errors
16566 included in this list under circumstances other than those described here, or may contain
16567 extensions or limitations that prevent some errors from occurring.

16568 The ERRORS section on each reference page specifies which error conditions shall be detected
16569 by all implementations ("shall fail") and which may be optionally detected by an
16570 implementation ("may fail"). If no error condition is detected, the action requested shall be
16571 successful. If an error condition is detected, the action requested may have been partially
16572 performed, unless otherwise stated.

16573 Implementations may generate error numbers listed here under circumstances other than those
16574 described, if and only if all those error conditions can always be treated identically to the error
16575 conditions as described in this volume of POSIX.1-2017. Implementations shall not generate a
16576 different error number from one required by this volume of POSIX.1-2017 for an error condition
16577 described in this volume of POSIX.1-2017, but may generate additional errors unless explicitly
16578 disallowed for a particular function.

16579 Each implementation shall document, in the conformance document, situations in which each of
16580 the optional conditions defined in POSIX.1-2017 is detected. The conformance document may
16581 also contain statements that one or more of the optional error conditions are not detected.

16582 Certain threads-related functions are not allowed to return an error code of [EINTR]. Where this
16583 applies it is stated in the ERRORS section on the individual function pages.

16584 The following macro names identify the possible error numbers, in the context of the functions
16585 specifically defined in this volume of POSIX.1-2017; these general descriptions are more
16586 precisely defined in the ERRORS sections of the functions that return them. Only these macro
16587 names should be used in programs, since the actual value of the error number is unspecified. All
16588 values listed in this section shall be unique, except as noted below. The values for all these
16589 macros shall be found in the `<errno.h>` header defined in the Base Definitions volume of
16590 POSIX.1-2017. The actual values are unspecified by this volume of POSIX.1-2017.

16591 [E2BIG]

16592 Argument list too long. The sum of the number of bytes used by the new process image's
16593 argument list and environment list is greater than the system-imposed limit of {ARG_MAX}
16594 bytes.

16595 or:

16596 Lack of space in an output buffer.

16597 or:

16598		Argument is greater than the system-imposed maximum.
16599		[EACCES]
16600		Permission denied. An attempt was made to access a file in a way forbidden by its file
16601		access permissions.
16602		[EADDRINUSE]
16603		Address in use. The specified address is in use.
16604		[EADDRNOTAVAIL]
16605		Address not available. The specified address is not available from the local system.
16606		[EAFNOSUPPORT]
16607		Address family not supported. The implementation does not support the specified address
16608		family, or the specified address is not a valid address for the address family of the specified
16609		socket.
16610		[EAGAIN]
16611		Resource temporarily unavailable. This is a temporary condition and later calls to the same
16612		routine may complete normally.
16613		[EALREADY]
16614		Connection already in progress. A connection request is already in progress for the specified
16615		socket.
16616		[EBADF]
16617		Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
16618		read (write) request is made to a file that is only open for writing (reading).
16619		[EBADMSG]
16620	OB XSR	Bad message. During a <i>read()</i> , <i>getmsg()</i> , <i>getpmsg()</i> , or <i>ioctl()</i> I_RECVFD request to a
16621		STREAMS device, a message arrived at the head of the STREAM that is inappropriate for
16622		the function receiving the message.
16623		<i>read()</i> Message waiting to be read on a STREAM is not a data message.
16624		<i>getmsg()</i> or <i>getpmsg()</i>
16625		A file descriptor was received instead of a control message.
16626		<i>ioctl()</i> Control or data information was received instead of a file descriptor when
16627		I_RECVFD was specified.
16628		or:
16629		Bad Message. The implementation has detected a corrupted message.
16630		[EBUSY]
16631		Resource busy. An attempt was made to make use of a system resource that is not currently
16632		available, as it is being used by another process in a manner that would have conflicted
16633		with the request being made by this process.
16634		[ECANCELED]
16635		Operation canceled. The associated asynchronous operation was canceled before
16636		completion.
16637		[ECHILD]
16638		No child process. A <i>wait()</i> , <i>waitid()</i> , or <i>waitpid()</i> function was executed by a process that
16639		had no existing or unwaited-for child process.

16640	[ECONNABORTED]
16641	Connection aborted. The connection has been aborted.
16642	[ECONNREFUSED]
16643	Connection refused. An attempt to connect to a socket was refused because there was no
16644	process listening or because the queue of connection requests was full and the underlying
16645	protocol does not support retransmissions.
16646	[ECONNRESET]
16647	Connection reset. The connection was forcibly closed by the peer.
16648	[EDEADLK]
16649	Resource deadlock would occur. An attempt was made to lock a system resource that would
16650	have resulted in a deadlock situation.
16651	[EDESTADDRREQ]
16652	Destination address required. No bind address was established.
16653	[EDOM]
16654	Domain error. An input argument is outside the defined domain of the mathematical
16655	function (defined in the ISO C standard).
16656	[EDQUOT]
16657	Reserved.
16658	[EEXIST]
16659	File exists. An existing file was mentioned in an inappropriate context; for example, as a
16660	new link name in the <i>link()</i> function.
16661	[EFAULT]
16662	Bad address. The system detected an invalid address in attempting to use an argument of a
16663	call. The reliable detection of this error cannot be guaranteed, and when not detected may
16664	result in the generation of a signal, indicating an address violation, which is sent to the
16665	process.
16666	[EFBIG]
16667	File too large. The size of a file would exceed the maximum file size of an implementation
16668	or offset maximum established in the corresponding file description.
16669	[EHOSTUNREACH]
16670	Host is unreachable. The destination host cannot be reached (probably because the host is
16671	down or a remote router cannot reach it).
16672	[EIDRM]
16673	Identifier removed. Returned during XSI interprocess communication if an identifier has
16674	been removed from the system.
16675	[EILSEQ]
16676	Illegal byte sequence. A wide-character code has been detected that does not correspond to
16677	a valid character, or a byte sequence does not form a valid wide-character code (defined in
16678	the ISO C standard).
16679	[EINPROGRESS]
16680	Operation in progress. This code is used to indicate that an asynchronous operation has not
16681	yet completed.
16682	or:
16683	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
16684	immediately established.

16685	[EINTR]
16686	Interrupted function call. An asynchronous signal was caught by the process during the
16687	execution of an interruptible function. If the signal handler performs a normal return, the
16688	interrupted function call may return this condition (see the Base Definitions volume of
16689	POSIX.1-2017, <signal.h>).
16690	[EINVAL]
16691	Invalid argument. Some invalid argument was supplied; for example, specifying an
16692	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.
16693	[EIO]
16694	Input/output error. Some physical input or output error has occurred. This error may be
16695	reported on a subsequent operation on the same file descriptor. Any other error-causing
16696	operation on the same file descriptor may cause the [EIO] error indication to be lost.
16697	[EISCONN]
16698	Socket is connected. The specified socket is already connected.
16699	[EISDIR]
16700	Is a directory. An attempt was made to open a directory with write mode specified.
16701	[ELOOP]
16702	Symbolic link loop. A loop exists in symbolic links encountered during pathname
16703	resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
16704	are encountered during pathname resolution.
16705	[EMFILE]
16706	File descriptor value too large or too many open streams. An attempt was made to open a
16707	file descriptor with a value greater than or equal to {OPEN_MAX}, or greater than or equal
16708	to the soft limit RLIMIT_NOFILE for the process (if smaller than {OPEN_MAX}); or an
16709	attempt was made to open more than the maximum number of streams allowed in the
16710	process.
16711	[EMLINK]
16712	Too many links. An attempt was made to have the link count of a single file exceed
16713	{LINK_MAX}.
16714	[EMSGSIZE]
16715	Message too large. A message sent on a transport provider was larger than an internal
16716	message buffer or some other network limit.
16717	or:
16718	Inappropriate message buffer length.
16719	[EMULTIHOP]
16720	Reserved.
16721	[ENAMETOOLONG]
16722	Filename too long. The length of a pathname exceeds {PATH_MAX} and the
16723	implementation considers this to be an error, or a pathname component is longer than
16724	{NAME_MAX}. This error may also occur when pathname substitution, as a result of
16725	encountering a symbolic link during pathname resolution, results in a pathname string the
16726	size of which exceeds {PATH_MAX}.
16727	[ENETDOWN]
16728	Network is down. The local network interface used to reach the destination is down.

16729		[ENETRESET]
16730		The connection was aborted by the network.
16731		[ENETUNREACH]
16732		Network unreachable. No route to the network is present.
16733		[ENFILE]
16734		Too many files open in system. Too many files are currently open in the system. The system
16735		has reached its predefined limit for simultaneously open files and temporarily cannot
16736		accept requests to open another one.
16737		[ENOBUFS]
16738		No buffer space available. Insufficient buffer resources were available in the system to
16739		perform the socket operation.
16740	OB XSR	[ENODATA]
16741		No message available. No message is available on the STREAM head read queue.
16742		[ENODEV]
16743		No such device. An attempt was made to apply an inappropriate function to a device; for
16744		example, trying to read a write-only device such as a printer.
16745		[ENOENT]
16746		No such file or directory. A component of a specified pathname does not exist, or the
16747		pathname is an empty string.
16748		[ENOEXEC]
16749		Executable file format error. A request is made to execute a file that, although it has
16750		appropriate privileges, is not in the format required by the implementation for executable
16751		files.
16752		[ENOLCK]
16753		No locks available. A system-imposed limit on the number of simultaneous file and record
16754		locks has been reached and no more are currently available.
16755		[ENOLINK]
16756		Reserved.
16757		[ENOMEM]
16758		Not enough space. The new process image requires more memory than is allowed by the
16759		hardware or system-imposed memory management constraints.
16760		[ENOMSG]
16761		No message of the desired type. The message queue does not contain a message of the
16762		required type during XSI interprocess communication.
16763		[ENOPROTOPT]
16764		Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the
16765		implementation.
16766		[ENOSPC]
16767		No space left on a device. During the <i>write()</i> function on a regular file or when extending a
16768		directory, there is no free space left on the device.
16769	OB XSR	[ENOSR]
16770		No STREAM resources. Insufficient STREAMS memory resources are available to perform a
16771		STREAMS-related function. This is a temporary condition; it may be recovered from if other
16772		processes release resources.

16773	OB XSR	[ENOSTR]
16774		Not a STREAM. A STREAM function was attempted on a file descriptor that was not
16775		associated with a STREAMS device.
16776		[ENOSYS]
16777		Functionality not supported. An attempt was made to use optional functionality that is not
16778		supported in this implementation.
16779		[ENOTCONN]
16780		Socket not connected. The socket is not connected.
16781		[ENOTDIR]
16782		Not a directory. A component of the specified pathname exists, but it is not a directory,
16783		when a directory was expected; or an attempt was made to create a non-directory file, and
16784		the specified pathname contains at least one non- <code><slash></code> character and ends with one or
16785		more trailing <code><slash></code> characters.
16786		[ENOTEMPTY]
16787		Directory not empty. A directory other than an empty directory was supplied when an
16788		empty directory was expected.
16789		[ENOTRECOVERABLE]
16790		State not recoverable. The state protected by a robust mutex is not recoverable.
16791		[ENOTSOCK]
16792		Not a socket. The file descriptor does not refer to a socket.
16793		[ENOTSUP]
16794		Not supported. The implementation does not support the requested feature or value.
16795		[ENOTTY]
16796		Inappropriate I/O control operation. A control function has been attempted for a file or
16797		special file for which the operation is inappropriate.
16798		[ENXIO]
16799		No such device or address. Input or output on a special file refers to a device that does not
16800		exist, or makes a request beyond the capabilities of the device. It may also occur when, for
16801		example, a tape drive is not on-line.
16802		[EOPNOTSUPP]
16803		Operation not supported on socket. The type of socket (address family or protocol) does not
16804		support the requested operation. A conforming implementation may assign the same values
16805		for [EOPNOTSUPP] and [ENOTSUP].
16806		[EOVERFLOW]
16807		Value too large to be stored in data type. An operation was attempted which would
16808		generate a value that is outside the range of values that can be represented in the relevant
16809		data type or that are allowed for a given data item.
16810		[EOWNERDEAD]
16811		Previous owner died. The owner of a robust mutex terminated while holding the mutex
16812		lock.
16813		[EPERM]
16814		Operation not permitted. An attempt was made to perform an operation limited to
16815		processes with appropriate privileges or to the owner of a file or other resource.
16816		[EPIPE]
16817		Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process
16818		to read the data.

16819	[EPROTO]	
16820		Protocol error. Some protocol error occurred. This error is device-specific, but is generally
16821		not related to a hardware failure.
16822	[EPROTONOSUPPORT]	
16823		Protocol not supported. The protocol is not supported by the address family, or the protocol
16824		is not supported by the implementation.
16825	[EPROTOTYPE]	
16826		Protocol wrong type for socket. The socket type is not supported by the protocol.
16827	[ERANGE]	
16828		Result too large or too small. The result of the function is too large (overflow) or too small
16829		(underflow) to be represented in the available space (defined in the ISO C standard).
16830	[EROFS]	
16831		Read-only file system. An attempt was made to modify a file or directory on a file system
16832		that is read-only.
16833	[ESPIPE]	
16834		Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.
16835	[ESRCH]	
16836		No such process. No process can be found corresponding to that specified by the given
16837		process ID.
16838	[ESTALE]	
16839		Reserved.
16840	OB XSR [ETIME]	
16841		STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of
16842		this error is device-specific and could indicate either a hardware or software failure, or a
16843		timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation is
16844		unspecified.
16845	[ETIMEDOUT]	
16846		Connection timed out. The connection to a remote machine has timed out. If the connection
16847		timed out during execution of the function that reported this error (as opposed to timing
16848		out prior to the function being called), it is unspecified whether the function has completed
16849		some or all of the documented behavior associated with a successful completion of the
16850		function.
16851		or:
16852		Operation timed out. The time limit associated with the operation was exceeded before the
16853		operation completed.
16854	[ETXTBSY]	
16855		Text file busy. An attempt was made to execute a pure-procedure program that is currently
16856		open for writing, or an attempt has been made to open for writing a pure-procedure
16857		program that is being executed.
16858	[EWOULDBLOCK]	
16859		Operation would block. An operation on a socket marked as non-blocking has encountered
16860		a situation such as no data available that otherwise would have caused the function to
16861		suspend execution.
16862		A conforming implementation may assign the same values for [EWOULDBLOCK] and
16863		[EAGAIN].

16864 [EXDEV]
 16865 Improper link. A link to a file on another file system was attempted.

16866 2.3.1 Additional Error Numbers

16867 Additional implementation-defined error numbers may be defined in `<errno.h>`.

16868 2.4 Signal Concepts

16869 2.4.1 Signal Generation and Delivery

16870 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes
 16871 the signal first occurs. Examples of such events include detection of hardware faults, timer
 16872 expiration, signals generated via the `sigevent` structure and terminal activity, as well as
 16873 invocations of the `kill()` and `sigqueue()` functions. In some circumstances, the same event
 16874 generates signals for multiple processes.

16875 At the time of generation, a determination shall be made whether the signal has been generated
 16876 for the process or for a specific thread within the process. Signals which are generated by some
 16877 action attributable to a particular thread, such as a hardware fault, shall be generated for the
 16878 thread that caused the signal to be generated. Signals that are generated in association with a
 16879 process ID or process group ID or an asynchronous event, such as terminal activity, shall be
 16880 generated for the process.

16881 Each process has an action to be taken in response to each signal defined by the system (see
 16882 [Section 2.4.3](#)). A signal is said to be “delivered” to a process when the appropriate action for the
 16883 process and signal is taken. A signal is said to be “accepted” by a process when the signal is
 16884 selected and returned by one of the `sigwait()` functions.

16885 During the time between the generation of a signal and its delivery or acceptance, the signal is
 16886 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a
 16887 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal
 16888 is anything other than to ignore the signal, and if that signal is generated for the thread, the
 16889 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned
 16890 by a call to the `sigwait()` function, or the action associated with it is set to ignore the signal.
 16891 Signals generated for the process shall be delivered to exactly one of those threads within the
 16892 process which is in a call to a `sigwait()` function selecting that signal or has not blocked delivery
 16893 of the signal. If there are no threads in a call to a `sigwait()` function selecting that signal, and if all
 16894 threads within the process block delivery of the signal, the signal shall remain pending on the
 16895 process until a thread calls a `sigwait()` function selecting that signal, a thread unblocks delivery
 16896 of the signal, or the action associated with the signal is set to ignore the signal. If the action
 16897 associated with a blocked signal is to ignore the signal and if that signal is generated for the
 16898 process, it is unspecified whether the signal is discarded immediately upon generation or
 16899 remains pending.

16900 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery
 16901 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,
 16902 or from the corresponding thread in the parent process if the thread was created as the result of a
 16903 call to `fork()`. The `pthread_sigmask()`, `sigaction()`, `sigprocmask()`, and `sigsuspend()` functions control
 16904 the manipulation of the signal mask.

16905 The determination of which action is taken in response to a signal is made at the time the signal
 16906 is delivered, allowing for any changes since the time of generation. This determination is
 16907 independent of the means by which the signal was originally generated. If a subsequent
 16908 occurrence of a pending signal is generated, it is implementation-defined as to whether the
 16909 signal is delivered or accepted more than once in circumstances other than those in which
 16910 queuing is required. The order in which multiple, simultaneously pending signals outside the
 16911 range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

16912 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process or
 16913 thread, all pending SIGCONT signals for that process or any of the threads within that process
 16914 shall be discarded. Conversely, when SIGCONT is generated for a process or thread, all pending
 16915 stop signals for that process or any of the threads within that process shall be discarded. When
 16916 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the
 16917 SIGCONT signal is ignored by the process or is blocked by all threads within the process and
 16918 there are no threads in a call to a *sigwait()* function selecting SIGCONT. If SIGCONT is blocked
 16919 by all threads within the process, there are no threads in a call to a *sigwait()* function selecting
 16920 SIGCONT, and SIGCONT is not ignored by the process, the SIGCONT signal shall remain
 16921 pending on the process until it is either unblocked by a thread or a thread calls a *sigwait()*
 16922 function selecting SIGCONT, or a stop signal is generated for the process or any of the threads
 16923 within the process.

16924 An implementation shall document any condition not specified by this volume of POSIX.1-2017
 16925 under which the implementation generates signals.

16926 2.4.2 Realtime Signal Generation and Delivery

16927 This section describes functionality to support realtime signal generation and delivery.

16928 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O
 16929 completion, interprocess message arrival, and the *sigqueue()* function, support the specification
 16930 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**
 16931 structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the
 16932 following members:

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union signal	<i>sigev_value</i>	Signal value.
void*(*) (union signal)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

16933 The *sigev_notify* member specifies the notification mechanism to use when an asynchronous
 16934 event occurs. This volume of POSIX.1-2017 defines the following values for the *sigev_notify*
 16935 member:

16942 SIGEV_NONE No asynchronous notification shall be delivered when the event of
 16943 interest occurs.

16944 SIGEV_SIGNAL The signal specified in *sigev_signo* shall be generated for the process when
 16945 the event of interest occurs. If the implementation supports the Realtime
 16946 Signals Extension option and if the SA_SIGINFO flag is set for that signal
 16947 number, then the signal shall be queued to the process and the value
 16948 specified in *sigev_value* shall be the *si_value* component of the generated
 16949 signal. If SA_SIGINFO is not set for that signal number, it is unspecified
 16950 whether the signal is queued and what value, if any, is sent.

16951 SIGEV_THREAD A notification function shall be called to perform notification.

16952 An implementation may define additional notification mechanisms.

16953 The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the
16954 application-defined value to be passed to the signal-catching function at the time of the signal
16955 delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

16956 The **signal** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	<i>sival_int</i>	Integer signal value.
void*	<i>sival_ptr</i>	Pointer signal value.

16960 The *sival_int* member shall be used when the application-defined value is of type **int**; the
16961 *sival_ptr* member shall be used when the application-defined value is a pointer.

16962 When a signal is generated by the *sigqueue()* function or any signal-generating function that
16963 supports the specification of an application-defined value, the signal shall be marked pending
16964 and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along
16965 with the application-specified signal value. Multiple occurrences of signals so generated are
16966 queued in FIFO order. It is unspecified whether signals so generated are queued when the
16967 SA_SIGINFO flag is not set for that signal.

16968 Signals generated by the *kill()* function or other events that cause signals to occur, such as
16969 detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the
16970 implementation does not support queuing, shall have no effect on signals already queued for the
16971 same signal number.

16972 When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the
16973 behavior shall be as if the implementation delivers the pending unblocked signal with the
16974 lowest signal number within that range. No other ordering of signal delivery is specified.

16975 If, when a pending signal is delivered, there are additional signals queued to that signal number,
16976 the signal shall remain pending. Otherwise, the pending indication shall be reset.

16977 Multi-threaded programs can use an alternate event notification mechanism. When a
16978 notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value
16979 SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

16980 The function shall be executed in an environment as if it were the *start_routine* for a newly
16981 created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes*
16982 is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to
16983 PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute
16984 of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this
16985 thread is implementation-defined.

16986 2.4.3 Signal Actions

16987 There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a
16988 pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of
16989 the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows.

16990 **SIG_DFL**

16991 Signal-specific default action.

16992 The default actions for the signals defined in this volume of POSIX.1-2017 are specified under
16993 <**signal.h**>. The default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX
16994 shall be to terminate the process abnormally.

16995 If the default action is to terminate the process abnormally, the process is terminated as if by a
16996 call to `_exit()`, except that the status made available to `wait()`, `waitid()`, and `waitpid()` indicates
16997 XSI abnormal termination by the signal. If the default action is to terminate the process abnormally
16998 with additional actions, implementation-defined abnormal termination actions, such as creation
16999 of a core file, may also occur.

17000 If the default action is to stop the process, the execution of that process is temporarily
17001 suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process,
17002 unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any
17003 additional signals that are sent to the process shall not be delivered until the process is
17004 continued, except SIGKILL which always terminates the receiving process. A process that is a
17005 member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP,
17006 SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a
17007 process, the signal shall be discarded.

17008 If the default action is to ignore the signal, delivery of the signal shall have no effect on the
17009 process.

17010 Setting a signal action to SIG_DFL for a signal that is pending, and whose default action is to
17011 ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded,
17012 whether or not it is blocked. Any queued values pending shall be discarded and the resources
17013 used to queue them shall be released and returned to the system for other use.

17014 The default action for SIGCONT is to resume execution at the point where the process was
17015 stopped, after first handling any pending unblocked signals.

17016 XSI When a stopped process is continued, a SIGCHLD signal may be generated for its parent
17017 process, unless the parent process has set the SA_NOCLDSTOP flag.

17018 **SIG_IGN**

17019 Ignore signal.

17020 Delivery of the signal shall have no effect on the process. The behavior of a process is undefined
17021 after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by `kill()`,
17022 `sigqueue()`, or `raise()`.

17023 The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG_IGN.

17024 Setting a signal action to SIG_IGN for a signal that is pending shall cause the pending signal to
17025 be discarded, whether or not it is blocked.

17026 If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified,
17027 XSI except as specified under “Consequences of Process Termination” in the description of the
17028 `_Exit()` function (see XSH `_Exit()`, on page 553).

17029 Any queued values pending shall be discarded and the resources used to queue them shall be
17030 released and made available to queue other signals.

17031 **Pointer to a Function**

17032 Catch signal.

17033 On delivery of the signal, the receiving process is to execute the signal-catching function at the
 17034 specified address. After returning from the signal-catching function, the receiving process shall
 17035 resume execution at the point at which it was interrupted.

17036 If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as
 17037 a C-language function call as follows:

17038

```
void func(int signo);
```

17039 If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-
 17040 language function call as follows:

17041

```
void func(int signo, siginfo_t *info, void *context);
```

17042 where *func* is the specified signal-catching function, *signo* is the signal number of the signal
 17043 being delivered, and *info* is a pointer to a **siginfo_t** structure defined in **<signal.h>** containing at
 17044 least the following members:

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number.
int	<i>si_code</i>	Cause of the signal.
pid_t	<i>si_pid</i>	Sending process ID.
uid_t	<i>si_uid</i>	Real user ID of sending process.
void *	<i>si_addr</i>	Address of faulting instruction.
int	<i>si_status</i>	Exit value or signal.
union signal	<i>si_value</i>	Signal value.

17053 The *si_signo* member shall contain the signal number. This shall be the same as the *signo*
 17054 parameter. The *si_code* member shall contain a code identifying the cause of the signal. The
 17055 following non-signal-specific values are defined for *si_code*:

17056 SI_USER The signal was sent by the *kill()* function. The implementation may set *si_code*
 17057 to SI_USER if the signal was sent by the *raise()* or *abort()* functions or any
 17058 similar functions provided as implementation extensions.

17059 SI_QUEUE The signal was sent by the *sigqueue()* function.

17060 SI_TIMER The signal was generated by the expiration of a timer set by *timer_settime()*.

17061 SI_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

17062 MSG SI_MESGQ The signal was generated by the arrival of a message on an empty message
 17063 queue.

17064 Signal-specific values for *si_code* are also defined, as described in XBD **<signal.h>**.

17065 If the signal was not generated by one of the functions or events listed above, *si_code* shall be set
 17066 either to one of the signal-specific values described in XBD **<signal.h>**, or to an implementation-
 17067 defined value that is not equal to any of the values defined above.

17068 XSI If *si_code* is SI_USER or SI_QUEUE, or any value less than or equal to 0, then the signal was
 17069 generated by a process and *si_pid* and *si_uid* shall be set to the process ID and the real user ID of
 17070 the sender, respectively.

17071 In addition, *si_addr*, *si_pid*, *si_status*, and *si_uid* shall be set for certain signal-specific values of
 17072 *si_code*, as described in XBD **<signal.h>**.

17073 If *si_code* is one of SI_QUEUE, SI_TIMER, SI_ASYNCIO, or SI_MESGQ, then *si_value* shall

17074 contain the application-specified signal value. Otherwise, the contents of *si_value* are undefined.

17075 The behavior of a process is undefined after it returns normally from a signal-catching function
17076 for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not generated by *kill()*, *sigqueue()*,
17077 or *raise()*.

17078 The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

17079 If a process establishes a signal-catching function for the SIGCHLD signal while it has a
17080 terminated child process for which it has not waited, it is unspecified whether a SIGCHLD
17081 signal is generated to indicate that child process.

17082 If the process is multi-threaded, or if the process is single-threaded and a signal handler is
17083 executed other than as the result of:

17084 The process calling *abort()*, *raise()*, *kill()*, *pthread_kill()*, or *sigqueue()* to generate a signal
17085 that is not blocked

17086 A pending signal being unblocked and being delivered before the call that unblocked it
17087 returns

17088 the behavior is undefined if the signal handler refers to any object other than *errno* with static
17089 storage duration other than by assigning a value to an object declared as **volatile sig_atomic_t**,
17090 or if the signal handler calls any function defined in this standard other than one of the functions
17091 listed in the following table.

17092 The following table defines a set of functions that shall be async-signal-safe. Therefore,
17093 applications can call them, without restriction, from signal-catching functions. Note that,
17094 although there is no restriction on the calls themselves, for certain functions there are restrictions
17095 on subsequent behavior after the function is called from a signal-catching function (see
17096 [longjmp\(\)](#)).

17097	<code>_Exit()</code>	<code>getppid()</code>	<code>sendmsg()</code>	<code>tcgetpgrp()</code>
17098	<code>_exit()</code>	<code>getsockname()</code>	<code>sendto()</code>	<code>tcsendbreak()</code>
17099	<code>abort()</code>	<code>getsockopt()</code>	<code>setgid()</code>	<code>tcsetattr()</code>
17100	<code>accept()</code>	<code>getuid()</code>	<code>setpgid()</code>	<code>tcsetpgrp()</code>
17101	<code>access()</code>	<code>htonl()</code>	<code>setsid()</code>	<code>time()</code>
17102	<code>aio_error()</code>	<code>htons()</code>	<code>setsockopt()</code>	<code>timer_getoverrun()</code>
17103	<code>aio_return()</code>	<code>kill()</code>	<code>setuid()</code>	<code>timer_gettime()</code>
17104	<code>aio_suspend()</code>	<code>link()</code>	<code>shutdown()</code>	<code>timer_settime()</code>
17105	<code>alarm()</code>	<code>linkat()</code>	<code>sigaction()</code>	<code>times()</code>
17106	<code>bind()</code>	<code>listen()</code>	<code>sigaddset()</code>	<code>umask()</code>
17107	<code>cfgetispeed()</code>	<code>longjmp()</code>	<code>sigdelset()</code>	<code>uname()</code>
17108	<code>cfgetospeed()</code>	<code>lseek()</code>	<code>sigemptyset()</code>	<code>unlink()</code>
17109	<code>cfsetispeed()</code>	<code>lstat()</code>	<code>sigfillset()</code>	<code>unlinkat()</code>
17110	<code>cfsetospeed()</code>	<code>memccpy()</code>	<code>sigismember()</code>	<code>utime()</code>
17111	<code>chdir()</code>	<code>memchr()</code>	<code>siglongjmp()</code>	<code>utimensat()</code>
17112	<code>chmod()</code>	<code>memcmp()</code>	<code>signal()</code>	<code>utimes()</code>
17113	<code>chown()</code>	<code>memcpy()</code>	<code>sigpause()</code>	<code>wait()</code>
17114	<code>clock_gettime()</code>	<code>memmove()</code>	<code>sigpending()</code>	<code>waitpid()</code>
17115	<code>close()</code>	<code>memset()</code>	<code>sigprocmask()</code>	<code>wcpcpy()</code>
17116	<code>connect()</code>	<code>mkdir()</code>	<code>sigqueue()</code>	<code>wcpncpy()</code>
17117	<code>creat()</code>	<code>mkdirat()</code>	<code>sigset()</code>	<code>wscat()</code>
17118	<code>dup()</code>	<code>mkfifo()</code>	<code>sigsuspend()</code>	<code>wcschr()</code>
17119	<code>dup2()</code>	<code>mkfifoat()</code>	<code>sleep()</code>	<code>wcscmp()</code>
17120	<code>execl()</code>	<code>mknod()</code>	<code>socketatmark()</code>	<code>wcscpy()</code>
17121	<code>execle()</code>	<code>mknodat()</code>	<code>socket()</code>	<code>wcscspn()</code>
17122	<code>execv()</code>	<code>ntohl()</code>	<code>socketpair()</code>	<code>wcslen()</code>
17123	<code>execve()</code>	<code>ntohs()</code>	<code>stat()</code>	<code>wcsncat()</code>
17124	<code>faccessat()</code>	<code>open()</code>	<code>stpncpy()</code>	<code>wcsncmp()</code>
17125	<code>fchdir()</code>	<code>openat()</code>	<code>stpncpy()</code>	<code>wcsncpy()</code>
17126	<code>fchmod()</code>	<code>pause()</code>	<code>strcat()</code>	<code>wcsnlen()</code>
17127	<code>fchmodat()</code>	<code>pipe()</code>	<code>strchr()</code>	<code>wcspbrk()</code>
17128	<code>fchown()</code>	<code>poll()</code>	<code>strcmp()</code>	<code>wcsrchr()</code>
17129	<code>fchownat()</code>	<code>posix_trace_event()</code>	<code>strcpy()</code>	<code>wcsspn()</code>
17130	<code>fcntl()</code>	<code>pselect()</code>	<code>strcspn()</code>	<code>wcsstr()</code>
17131	<code>fdatasync()</code>	<code>pthread_kill()</code>	<code>strlen()</code>	<code>wcstok()</code>
17132	<code>fexecve()</code>	<code>pthread_self()</code>	<code>strncat()</code>	<code>wmemchr()</code>
17133	<code>ffs()</code>	<code>pthread_sigmask()</code>	<code>strncmp()</code>	<code>wmemcmp()</code>
17134	<code>fork()</code>	<code>raise()</code>	<code>strncpy()</code>	<code>wmemcpy()</code>
17135	<code>fstat()</code>	<code>read()</code>	<code>strnlen()</code>	<code>wmemmove()</code>
17136	<code>fstatat()</code>	<code>readlink()</code>	<code>strpbrk()</code>	<code>wmemset()</code>
17137	<code>fsync()</code>	<code>readlinkat()</code>	<code>strrchr()</code>	<code>write()</code>
17138	<code>fruncate()</code>	<code>recv()</code>	<code>strspn()</code>	
17139	<code>futimens()</code>	<code>recvfrom()</code>	<code>strstr()</code>	
17140	<code>getegid()</code>	<code>recvmsg()</code>	<code>strtok_r()</code>	
17141	<code>geteuid()</code>	<code>rename()</code>	<code>symlink()</code>	
17142	<code>getgid()</code>	<code>renameat()</code>	<code>symlinkat()</code>	
17143	<code>getgroups()</code>	<code>rmdir()</code>	<code>tcdrain()</code>	
17144	<code>getpeername()</code>	<code>select()</code>	<code>tcflow()</code>	
17145	<code>getpgrp()</code>	<code>sem_post()</code>	<code>tcflush()</code>	
17146	<code>getpid()</code>	<code>send()</code>	<code>tcgetattr()</code>	

17147 Any function not in the above table may be unsafe with respect to signals. Implementations may
 17148 make other interfaces async-signal-safe. In the presence of signals, all functions defined by this

17149 volume of POSIX.1-2017 shall behave as defined when called from or interrupted by a signal-
 17150 catching function, with the exception that when a signal interrupts an unsafe function or
 17151 equivalent (such as the processing equivalent to *exit()* performed after a return from the initial
 17152 call to *main()*) and the signal-catching function calls an unsafe function, the behavior is
 17153 undefined. Additional exceptions are specified in the descriptions of individual functions such
 17154 as *longjmp()*.

17155 Operations which obtain the value of *errno* and operations which assign a value to *errno* shall be
 17156 async-signal-safe, provided that the signal-catching function saves the value of *errno* upon entry
 17157 and restores it before it returns.

17158 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or
 17159 continue, the entire process shall be terminated, stopped, or continued, respectively.

17160 2.4.4 Signal Effects on Other Functions

17161 Signals affect the behavior of certain functions defined by this volume of POSIX.1-2017 if
 17162 delivered to a process while it is executing such a function. If the action of the signal is to
 17163 terminate the process, the process shall be terminated and the function shall not return. If the
 17164 action of the signal is to stop the process, the process shall stop until continued or terminated.
 17165 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the
 17166 original function shall continue at the point the process was stopped. If the action of the signal is
 17167 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case
 17168 the original function is said to be “interrupted” by the signal. If the signal-catching function
 17169 executes a **return** statement, the behavior of the interrupted function shall be as described
 17170 individually for that function, except as noted for unsafe functions. After returning from a
 17171 signal-catching function, the value of *errno* is unspecified if the signal-catching function or any
 17172 function it called assigned a value to *errno* and the signal-catching function did not save and
 17173 restore the original value of *errno*. Signals that are ignored shall not affect the behavior of any
 17174 function; signals that are blocked shall not affect the behavior of any function until they are
 17175 unblocked and then delivered, except as specified for the *sigpending()* and *sigwait()* functions.

17176 2.5 Standard I/O Streams

17177 CX A stream is associated with an external file (which may be a physical device) or memory buffer
 17178 CX by “opening” a file or buffer. This may involve “creating” a new file. Creating an existing file
 17179 causes its former contents to be discarded if necessary. If a file can support positioning requests
 17180 (such as a disk file, as opposed to a terminal), then a “file position indicator” associated with the
 17181 stream is positioned at the start (byte number 0) of the file, unless the file is opened with append
 17182 mode, in which case it is implementation-defined whether the file position indicator is initially
 17183 positioned at the beginning or end of the file. The file position indicator is maintained by
 17184 subsequent reads, writes, and positioning requests, to facilitate an orderly progression through
 17185 the file.

17186 The wide-character input functions shall read characters from the stream and convert them to
 17187 wide characters as if they were read by successive calls to the *fgetwc()* function. Each conversion
 17188 shall occur as if by a call to the *mbrtowc()* function, with the conversion state described by the
 17189 stream’s own **mbstate_t** object (see Section 2.5.2, on page 498). The byte input functions shall
 17190 read characters from the stream as if by successive calls to the *fgetc()* function.

17191 The wide-character output functions shall convert wide characters to characters and write them
 17192 to the stream as if they were written by successive calls to the *fputwc()* function. Each conversion

17193 shall occur as if by a call to the *wcrtomb()* function, with the conversion state described by the
17194 stream's own **mbstate_t** object (see [Section 2.5.2](#), on page 498). The byte output functions shall
17195 write characters to the stream as if by successive calls to the *fputc()* function.

17196 The *perror()*, *psiginfo()*, and *psignal()* functions shall behave as described above for the byte
17197 output functions if the stream is already byte-oriented, and shall behave as described above for
17198 the wide-character output functions if the stream is already wide-oriented. If the stream has no
17199 orientation, they shall behave as described for the byte output functions except that they shall
17200 not change the orientation of the stream.

17201 Functions other than *perror()*, *psiginfo()*, and *psignal()* that write to streams but are neither wide-
17202 character output nor byte output functions (*getopt()* and *wordexp()*), shall behave as described
17203 above for the byte output functions, except that if the stream has no orientation, it is unspecified
17204 whether they set the stream to byte orientation or leave it with no orientation.

17205 When a stream is "unbuffered", bytes are intended to appear from the source or at the
17206 destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a
17207 block. When a stream is "fully buffered", bytes are intended to be transmitted as a block when a
17208 buffer is filled. When a stream is "line buffered", bytes are intended to be transmitted as a block
17209 when a <newline> byte is encountered. Furthermore, bytes are intended to be transmitted as a
17210 block when a buffer is filled, when input is requested on an unbuffered stream, or when input is
17211 requested on a line-buffered stream that requires the transmission of bytes. Support for these
17212 characteristics is implementation-defined, and may be affected via *setbuf()* and *setvbuf()*.

17213 A file may be disassociated from a controlling stream by "closing" the file. Output streams are
17214 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from
17215 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed
17216 (including the standard streams).

17217 A file may be subsequently reopened, by the same or another program execution, and its
17218 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function
17219 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all
17220 output streams are flushed) before program termination. Other paths to program termination,
17221 such as calling *abort()*, need not close all files properly.

17222 The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE**
17223 object need not necessarily serve in place of the original.

17224 At program start-up, three streams are predefined and need not be opened explicitly: *standard*
17225 *input* (for reading conventional input), *standard output* (for writing conventional output), and
17226 *standard error* (for writing diagnostic output). When opened, the standard error stream is not
17227 fully buffered; the standard input and standard output streams are fully buffered if and only if
17228 the stream can be determined not to refer to an interactive device.

17229 CX A stream associated with a memory buffer shall have the same operations for text files that a
17230 stream associated with an external file would have. In addition, the stream orientation shall be
17231 determined in exactly the same fashion.

17232 Input and output operations on a stream associated with a memory buffer by a call to
17233 *fmemopen()* shall be constrained by the implementation to take place within the bounds of the
17234 memory buffer. In the case of a stream opened by *open_memstream()* or *open_wmemstream()*, the
17235 memory area shall grow dynamically to accommodate write operations as necessary. For output,
17236 data is moved from the buffer provided by *setvbuf()* to the memory stream during a flush or
17237 close operation.

17238 2.5.1 Interaction of File Descriptors and Standard I/O Streams

17239 CX This section describes the interaction of file descriptors and standard I/O streams. The
17240 functionality described in this section is an extension to the ISO C standard (and the rest of this
17241 section is not further CX shaded).

17242 An open file description may be accessed through a file descriptor, which is created using
17243 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as
17244 *fopen()* or *popen()*. Either a file descriptor or a stream is called a “handle” on the open file
17245 description to which it refers; an open file description may have several handles.

17246 Handles can be created or destroyed by explicit user action, without affecting the underlying
17247 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,
17248 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

17249 A file descriptor that is never used in an operation that could affect the file offset (for example,
17250 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to
17251 one (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include
17252 the file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is
17253 not used directly by the application to affect the file offset. The *read()* and *write()* functions
17254 implicitly affect the file offset; *lseek()* explicitly affects it.

17255 The result of function calls involving any one handle (the “active handle”) is defined elsewhere
17256 in this volume of POSIX.1-2017, but if two or more handles are used, and any one of them is a
17257 stream, the application shall ensure that their actions are coordinated as described below. If this
17258 is not done, the result is undefined.

17259 A handle which is a stream is considered to be closed when either an *fclose()*, or *freopen()* with
17260 non-full filename, is executed on it (for *freopen()* with a null filename, it is implementation-
17261 defined whether a new handle is created or the existing one reused), or when the process
17262 owning that stream terminates with *exit()*, *abort()*, or due to a signal. A file descriptor is closed
17263 by *close()*, *_exit()*, or the *exec* functions when *FD_CLOEXEC* is set on that file descriptor.

17264 For a handle to become the active handle, the application shall ensure that the actions below are
17265 performed between the last use of the handle (the current active handle) and the first use of the
17266 second handle (the future active handle). The second handle then becomes the active handle. All
17267 activity by the application affecting the file offset on the first handle shall be suspended until it
17268 again becomes the active file handle. (If a stream function has as an underlying function one that
17269 affects the file offset, the stream function shall be considered to affect the file offset.)

17270 The handles need not be in the same process for these rules to apply.

17271 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure
17272 that, if both handles can ever be accessed, they are both in a state where the other could become
17273 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of
17274 active handle. (If the only action performed by one of the processes is one of the *exec* functions or
17275 *_exit()* (not *exit()*), the handle is never accessed in that process.)

17276 For the first handle, the first applicable condition below applies. After the actions required
17277 below are taken, if the handle is still open, the application can close it.

17278 If it is a file descriptor, no action is required.

17279 If the only further action to be performed on any handle to this open file descriptor is to
17280 close it, no action need be taken.

17281 If it is a stream which is unbuffered, no action need be taken.

17282 If it is a stream which is line buffered, and the last byte written to the stream was a
17283 <newline> (that is, as if a:

```
17284         putchar('\n')
```

17285 was the most recent operation on that stream), no action need be taken.

17286 If it is a stream which is open for writing or appending (but not also open for reading), the
17287 application shall either perform an *fflush()*, or the stream shall be closed.

17288 If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need
17289 be taken.

17290 If the stream is open with a mode that allows reading and the underlying open file
17291 description refers to a device that is capable of seeking, the application shall either perform
17292 an *fflush()*, or the stream shall be closed.

17293 For the second handle:

17294 If any previous active handle has been used by a function that explicitly changed the file
17295 offset, except as required above for the first handle, the application shall perform an *lseek()*
17296 or *fseek()* (as appropriate to the type of handle) to an appropriate location.

17297 If the active handle ceases to be accessible before the requirements on the first handle, above,
17298 have been met, the state of the open file description becomes undefined. This might occur
17299 during functions such as a *fork()* or *_exit()*.

17300 The *exec* functions make inaccessible all streams that are open at the time they are called,
17301 independent of which streams or file descriptors may be available to the new process image.

17302 When these rules are followed, regardless of the sequence of handles used, implementations
17303 shall ensure that an application, even one consisting of several processes, shall yield correct
17304 results: no data shall be lost or duplicated when writing, and all data shall be written in order,
17305 except as requested by seeks. It is implementation-defined whether, and under what conditions,
17306 all input is seen exactly once.

17307 Each function that operates on a stream is said to have zero or more “underlying functions”.
17308 This means that the stream function shares certain traits with the underlying functions, but does
17309 not require that there be any relation between the implementations of the stream function and its
17310 underlying functions.

17311 2.5.2 Stream Orientation and Encoding Rules

17312 For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an
17313 “orientation”. After a stream is associated with an external file, but before any operations are
17314 performed on it, the stream is without orientation. Once a wide-character input/output function
17315 has been applied to a stream without orientation, the stream shall become “wide-oriented”.
17316 Similarly, once a byte input/output function has been applied to a stream without orientation,
17317 the stream shall become “byte-oriented”. Only a call to the *freopen()* function or the *fwide()*
17318 function can otherwise alter the orientation of a stream.

17319 A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard*
17320 *input*, *standard output*, and *standard error* shall be unoriented at program start-up.

17321 Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character
17322 input/output functions cannot be applied to a byte-oriented stream. The remaining stream
17323 operations shall not affect and shall not be affected by a stream’s orientation, except for the
17324 following additional restriction:

17325 For wide-oriented streams, after a successful call to a file-positioning function that leaves
 17326 the file position indicator prior to the end-of-file, a wide-character output function can
 17327 overwrite a partial character; any file contents beyond the byte(s) written are henceforth
 17328 undefined.

17329 Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state
 17330 of the stream. A successful call to *fgetpos()* shall store a representation of the value of this
 17331 **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos()* using
 17332 the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as
 17333 the position within the controlled stream.

17334 Implementations that support multiple encoding rules associate an encoding rule with the
 17335 stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the
 17336 current locale at the time when the stream becomes wide-oriented. As with the stream's
 17337 orientation, the encoding rule associated with a stream cannot be changed once it has been set,
 17338 except by a successful call to *freopen()* which clears the encoding rule and resets the orientation
 17339 to unoriented.

17340 Although wide-oriented streams are conceptually sequences of wide characters, the external file
 17341 associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters
 17342 generalized as follows:

17343 Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte
 17344 encodings valid for use internal to the program).

17345 A file need not begin nor end in the initial shift state.

17346 Moreover, the encodings used for characters may differ among files. Both the nature and choice
 17347 of such encodings are implementation-defined.

17348 The wide-character input functions read characters from the stream and convert them to wide
 17349 characters as if they were read by successive calls to the *fgetwc()* function. Each conversion shall
 17350 occur as if by a call to the *mbrtowc()* function, with the conversion state described by the
 17351 CX stream's own **mbstate_t** object, except the encoding rule associated with the stream is used
 17352 instead of the encoding rule implied by the *LC_CTYPE* category of the current locale.

17353 The wide-character output functions convert wide characters to (possibly multi-byte) characters
 17354 and write them to the stream as if they were written by successive calls to the *fputwc()* function.
 17355 Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state
 17356 CX described by the stream's own **mbstate_t** object, except the encoding rule associated with the
 17357 stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current
 17358 locale.

17359 An "encoding error" shall occur if the character sequence presented to the underlying *mbrtowc()*
 17360 function does not form a valid (generalized) character, or if the code value passed to the
 17361 underlying *wcrtomb()* function does not correspond to a valid (generalized) character. The wide-
 17362 character input/output functions and the byte input/output functions store the value of the
 17363 macro [EILSEQ] in *errno* if and only if an encoding error occurs.

17364 **2.6 STREAMS**

17365 OB XSR STREAMS functionality is provided on implementations supporting the XSI STREAMS Option
17366 Group. The functionality described in this section is dependent on support of the XSI STREAMS
17367 option (and the rest of this section is not further shaded for this option).

17368 STREAMS provides a uniform mechanism for implementing networking services and other
17369 character-based I/O. The STREAMS function provides direct access to protocol modules.
17370 STREAMS modules are unspecified objects. Access to STREAMS modules is provided by
17371 interfaces in POSIX.1-2017. Creation of STREAMS modules is outside the scope of
17372 POSIX.1-2017.

17373 A STREAM is typically a full-duplex connection between a process and an open device or
17374 pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex
17375 connection between two processes. The STREAM itself exists entirely within the implementation
17376 and provides a general character I/O function for processes. It optionally includes one or more
17377 intermediate processing modules that are interposed between the process end of the STREAM
17378 (STREAM head) and a device driver at the end of the STREAM (STREAM end).

17379 STREAMS I/O is based on messages. There are three types of message:

17380 *Data messages* containing actual data for input or output

17381 *Control data* containing instructions for the STREAMS modules and underlying
17382 implementation

17383 Other messages, which include file descriptors

17384 The interface between the STREAM and the rest of the implementation is provided by a set of
17385 functions at the STREAM head. When a process calls *write()*, *writew()*, *putmsg()*, *putpmsg()*, or
17386 *ioctl()*, messages are sent down the STREAM, and *read()*, *readv()*, *getmsg()*, or *getpmsg()* accepts
17387 data from the STREAM and passes it to a process. Data intended for the device at the
17388 downstream end of the STREAM is packaged into messages and sent downstream, while data
17389 and signals from the device are composed into messages by the device driver and sent upstream
17390 to the STREAM head.

17391 When a STREAMS-based device is opened, a STREAM shall be created that contains the
17392 STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an
17393 implementation, when a pipe is created, two STREAMS shall be created, each containing a
17394 STREAM head. Other modules are added to the STREAM using *ioctl()*. New modules are
17395 “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the
17396 STREAM was a push-down stack.

17397 **Priority**

17398 Message types are classified according to their queuing priority and may be *normal* (non-
17399 priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that
17400 determines its ordering when placed on a queue. Normal messages have a priority band of 0
17401 and shall always be placed at the end of the queue following all other messages in the queue.
17402 High-priority messages are always placed at the head of a queue, but shall be discarded if there
17403 is already a high-priority message in the queue. Their priority band shall be ignored; they are
17404 high-priority by virtue of their type. Priority messages have a priority band greater than 0.
17405 Priority messages are always placed after any messages of the same or higher priority. High-
17406 priority and priority messages are used to send control and data information outside the normal
17407 flow of control. By convention, high-priority messages shall not be affected by flow control.
17408 Normal and priority messages have separate flow controls.

17409 **Message Parts**

17410 A process may access STREAMS messages that contain a data part, control part, or both. The
 17411 data part is that information which is transmitted over the communication medium and the
 17412 control information is used by the local STREAMS modules. The other types of messages are
 17413 used between modules and are not accessible to processes. Messages containing only a data part
 17414 are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writv()*.
 17415 Messages containing a control part with or without a data part are accessible via calls to
 17416 *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

17417 **2.6.1 Accessing STREAMS**

17418 A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*,
 17419 *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable
 17420 function definitions for general properties and errors.

17421 Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor
 17422 *fildev*. The control functions may be performed by the STREAM head, a STREAMS module, or
 17423 the STREAMS driver for the STREAM.

17424 STREAMS modules and drivers can detect errors, sending an error message to the STREAM
 17425 head, thus causing subsequent functions to fail and set *errno* to the value specified in the
 17426 message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request
 17427 alone by sending a negative acknowledgement message to the STREAM head. This shall cause
 17428 just the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

17429 **2.7 XSI Interprocess Communication**

17430 XSI This section describes extensions to support interprocess communication. The functionality
 17431 described in this section shall be provided on implementations that support the XSI option (and
 17432 the rest of this section is not further shaded).

17433 The following message passing, semaphore, and shared memory services form an XSI
 17434 interprocess communication facility. Certain aspects of their operation are common, and are
 17435 defined as follows.

17436

17437

17438

17439

17440

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmat()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmctl()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmdt()</i>
<i>msgsnd()</i>		<i>shmget()</i>

17441 Another interprocess communication facility is provided by functions in the Realtime Option
 17442 Group; see [Section 2.8](#) (on page 503).

17443 2.7.1 IPC General Description

17444 Each individual shared memory segment, message queue, and semaphore set shall be identified
 17445 by a unique positive integer, called, respectively, a shared memory identifier, *shmid*, a semaphore
 17446 identifier, *semid*, and a message queue identifier, *msqid*. The identifiers shall be returned by calls
 17447 to *shmget()*, *semget()*, and *msgget()*, respectively.

17448 Associated with each identifier is a data structure which contains data related to the operations
 17449 which may be or may have been performed; see the Base Definitions volume of POSIX.1-2017,
 17450 <*sys/shm.h*>, <*sys/sem.h*>, and <*sys/msg.h*> for their descriptions.

17451 Each of the data structures contains both ownership information and an **ipc_perm** structure (see
 17452 the Base Definitions volume of POSIX.1-2017, <*sys/ipc.h*>) which are used in conjunction to
 17453 determine whether or not read/write (read/alter for semaphores) permissions should be
 17454 granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as
 17455 a bit field which determines the permissions.

17456 The values of the bits are given below in octal notation.

17457	Bit	Meaning
17458	0400	Read by user.
17459	0200	Write by user.
17460	0040	Read by group.
17461	0020	Write by group.
17462	0004	Read by others.
17463	0002	Write by others.

17464 The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which
 17465 service is being used. In each case, read and write/alter permissions shall be granted to a
 17466 process if one or more of the following are true ("*xxx*" is replaced by *shm*, *sem*, or *msg*, as
 17467 appropriate):

17468 The process has appropriate privileges.

17469 The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data
 17470 structure associated with the IPC identifier, and the appropriate bit of the *user* field in
 17471 *xxx_perm.mode* is set.

17472 The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the
 17473 effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data
 17474 structure associated with the IPC identifier, and the appropriate bit of the *group* field in
 17475 *xxx_perm.mode* is set.

17476 The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the
 17477 effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data
 17478 structure associated with the IPC identifier, but the appropriate bit of the *other* field in
 17479 *xxx_perm.mode* is set.

17480 Otherwise, the permission shall be denied.

17481 In addition to the **ipc_perm** structure, each associated data structure includes several **time_t**
 17482 fields for recording timestamps of particular operations. When an operation is described as
 17483 setting a timestamp to the current time, that particular timestamp member of the associated data
 17484 structure shall be set to the largest **time_t** value which is not greater than the current time.

17485 2.8 Realtime

17486 This section defines functions to support the source portability of applications with realtime
17487 requirements. The presence of some of these functions is dependent on support for
17488 implementation options described in the text.

17489 The specific functional areas included in this section and their scope include the following. Full
17490 definitions of these terms can be found in XBD [Chapter 3](#) (on page 33).

- 17491 Semaphores
- 17492 Process Memory Locking
- 17493 Memory Mapped Files and Shared Memory Objects
- 17494 Priority Scheduling
- 17495 Realtime Signal Extension
- 17496 Timers
- 17497 Interprocess Communication
- 17498 Synchronized Input and Output
- 17499 Asynchronous Input and Output

17500 All the realtime functions defined in this volume of POSIX.1-2017 are portable, although some of
17501 the numeric parameters used by an implementation may have hardware dependencies.

17502 2.8.1 Realtime Signals

17503 See [Section 2.4.2](#) (on page 489).

17504 2.8.2 Asynchronous I/O

17505 An asynchronous I/O control block structure **aiocb** is used in many asynchronous I/O
17506 functions. It is defined in the Base Definitions volume of POSIX.1-2017, [<aio.h>](#) and has at least
17507 the following members:

Member Type	Member Name	Description
int	<i>aio_fildes</i>	File descriptor.
off_t	<i>aio_offset</i>	File offset.
volatile void*	<i>aio_buf</i>	Location of buffer.
size_t	<i>aio_nbytes</i>	Length of transfer.
int	<i>aio_reqprio</i>	Request priority offset.
struct sigevent	<i>aio_sigevent</i>	Signal number and value.
int	<i>aio_lio_opcode</i>	Operation to be performed.

17516 The *aio_fildes* element is the file descriptor on which the asynchronous operation is performed.

17517 If `O_APPEND` is not set for the file descriptor *aio_fildes* and if *aio_fildes* is associated with a
17518 device that is capable of seeking, then the requested operation takes place at the absolute
17519 position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to the
17520 operation with an *offset* argument equal to *aio_offset* and a *whence* argument equal to `SEEK_SET`.
17521 If `O_APPEND` is set for the file descriptor, or if *aio_fildes* is associated with a device that is
17522 incapable of seeking, write operations append to the file in the same order as the calls were
17523 made, with the following exception: under implementation-defined circumstances, such as

17524 operation on a multi-processor or when requests of differing priorities are submitted at the same
17525 time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming
17526 application to determine whether this relaxation applies, all strictly conforming applications
17527 which rely on ordering of output shall be written in such a way that they will operate correctly if
17528 the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the
17529 value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same
17530 as the *nbyte* and *buf* arguments defined by *read()* and *write()*, respectively.

17531 If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then
17532 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation
17533 based on the current scheduling priority of the calling process. The *aio_reqprio* member can be
17534 used to lower (but not raise) the asynchronous I/O operation priority and is within the range
17535 zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and
17536 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O
17537 requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and
17538 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted
17539 by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is
17540 unspecified. The priority of an asynchronous request is computed as (process scheduling
17541 priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an
17542 indication of the desired order of execution of the request relative to other asynchronous I/O
17543 requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same
17544 priority to a character special file are processed by the underlying device in FIFO order; the
17545 order of processing of requests of the same priority issued to files that are not character special
17546 files is unspecified. Numerically higher priority values indicate requests of higher priority. The
17547 value of *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous
17548 I/O requests to the same file are blocked waiting for a resource required for that I/O operation,
17549 the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests
17550 are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is
17551 implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall
17552 define for which files I/O prioritization is supported.

17553 The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as
17554 specified in [Section 2.4.1](#) (on page 488). If *aio_sigevent.sigev_notify* is `SIGEV_NONE`, then no
17555 signal shall be posted upon I/O completion, but the error status for the operation and the return
17556 status for the operation shall be set appropriately.

17557 The *aio_lio_opcode* field is used only by the *lio_listio()* call. The *lio_listio()* call allows multiple
17558 asynchronous I/O operations to be submitted at a single time. The function takes as an
17559 argument an array of pointers to **aiocb** structures. Each **aiocb** structure indicates the operation to
17560 be performed (read or write) via the *aio_lio_opcode* field.

17561 The address of the **aiocb** structure is used as a handle for retrieving the error status and return
17562 status of the asynchronous operation while it is in progress.

17563 The **aiocb** structure and the data buffers associated with the asynchronous I/O operation are
17564 being used by the system for asynchronous I/O while, and only while, the error status of the
17565 asynchronous operation is equal to `[EINPROGRESS]`. Applications shall not modify the **aiocb**
17566 structure while the structure is being used by the system for asynchronous I/O.

17567 The return status of the asynchronous operation is the number of bytes transferred by the I/O
17568 operation. If the error status is set to indicate an error completion, then the return status is set to
17569 the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned.
17570 When the error status is not equal to `[EINPROGRESS]`, the return status shall reflect the return
17571 status of the corresponding synchronous operation.

17572 **2.8.3 Memory Management**17573 2.8.3.1 *Memory Locking*

17574 MLR Range memory locking operations are defined in terms of pages. Implementations may restrict
 17575 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,
 17576 is the value of the configurable system variable {PAGESIZE}. If an implementation has no
 17577 restrictions on size or alignment, it may specify a 1-byte page size.

17578 ML|MLR Memory locking guarantees the residence of portions of the address space. It is implementation-
 17579 defined whether locking memory guarantees fixed translation between virtual addresses (as
 17580 seen by the process) and physical addresses. Per-process memory locks are not inherited across a
 17581 *fork()*, and all memory locks owned by a process are unlocked upon *exec* or process termination.
 17582 Unmapping of an address range removes any memory locks established on that address range
 17583 by this process.

17584 2.8.3.2 *Memory Mapped Files*

17585 Range memory mapping operations are defined in terms of pages. Implementations may
 17586 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,
 17587 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has
 17588 no restrictions on size or alignment, it may specify a 1-byte page size.

17589 Memory mapped files provide a mechanism that allows a process to access files by directly
 17590 incorporating file data into its address space. Once a file is mapped into a process address space,
 17591 the data can be manipulated as memory. If more than one process maps a file, its contents are
 17592 shared among them. If the mappings allow shared write access, then data written into the
 17593 memory object through the address space of one process appears in the address spaces of all
 17594 processes that similarly map the same portion of the memory object.

17595 SHM Shared memory objects are named regions of storage that may be independent of the file system
 17596 and can be mapped into the address space of one or more processes to allow them to share the
 17597 associated memory.

17598 SHM An *unlink()* of a file or *shm_unlink()* of a shared memory object, while causing the removal of
 17599 the name, does not unmap any mappings established for the object. Once the name has been
 17600 removed, the contents of the memory object are preserved as long as it is referenced. The
 17601 memory object remains referenced as long as a process has the memory object open or has some
 17602 area of the memory object mapped.

17603 2.8.3.3 *Memory Protection*

17604 When an object is mapped, various application accesses to the mapped region may result in
 17605 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and
 17606 SIGSEGV is used to indicate a protection violation or misuse of an address:

17607 A mapping may be restricted to disallow some types of access.

17608 Write attempts to memory that was mapped without write access, or any access to
 17609 memory mapped PROT_NONE, shall result in a SIGSEGV signal.

17610 References to unmapped addresses shall result in a SIGSEGV signal.

17611 Reference to whole pages within the mapping, but beyond the current length of the object,
17612 shall result in a SIGBUS signal.

17613 The size of the object is unaffected by access beyond the end of the object (even if a
17614 SIGBUS is not generated).

17615 2.8.3.4 *Typed Memory Objects*

17616 TYM The functionality described in this section shall be provided on implementations that support
17617 the Typed Memory Objects option (and the rest of this section is not further shaded for this
17618 option).

17619 Implementations may support the Typed Memory Objects option independently of support for
17620 memory mapped files or shared memory objects. Typed memory objects are implementation-
17621 configurable named storage pools accessible from one or more processors in a system, each via
17622 one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid
17623 combination of a storage pool and a port is identified through a name that is defined at system
17624 configuration time, in an implementation-defined manner; the name may be independent of the
17625 file system. Using this name, a typed memory object can be opened and mapped into process
17626 address space. For a given storage pool and port, it is necessary to support both dynamic
17627 allocation from the pool as well as mapping at an application-supplied offset within the pool;
17628 when dynamic allocation has been performed, subsequent deallocation must be supported.
17629 Lastly, accessing typed memory objects from different ports requires a method for obtaining the
17630 offset and length of contiguous storage of a region of typed memory (dynamically allocated or
17631 not); this allows typed memory to be shared among processes and/or processors while being
17632 accessed from the desired port.

17633 2.8.4 **Process Scheduling**

17634 PS The functionality described in this section shall be provided on implementations that support
17635 the Process Scheduling option (and the rest of this section is not further shaded for this option).

17636 **Scheduling Policies**

17637 The scheduling semantics described in this volume of POSIX.1-2017 are defined in terms of a
17638 conceptual model that contains a set of thread lists. No implementation structures are
17639 necessarily implied by the use of this conceptual model. It is assumed that no time elapses
17640 during operations described using this model, and therefore no simultaneous operations are
17641 possible. This model discusses only processor scheduling for runnable threads, but it should be
17642 noted that greatly enhanced predictability of realtime applications results if the sequencing of
17643 other resources takes processor scheduling policy into account.

17644 There is, conceptually, one thread list for each priority. A runnable thread will be on the thread
17645 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty
17646 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose
17647 of a scheduling policy is to define the allowable operations on this set of lists (for example,
17648 moving threads between and within lists).

17649 The POSIX model treats a "process" as an aggregation of system resources, including one or
17650 more threads that may be scheduled by the operating system on the processor(s) it controls.
17651 Although a process has its own set of scheduling attributes, these have an indirect effect (if any)
17652 on the scheduling behavior of individual threads as described below.

17653 Each thread shall be controlled by an associated scheduling policy and priority. These
17654 parameters may be specified by explicit application execution of the *pthread_setschedparam()*

17655 function. Additionally, the scheduling parameters of a thread (but not its scheduling policy) may
17656 be changed by application execution of the *pthread_setschedprio()* function.

17657 Each process shall be controlled by an associated scheduling policy and priority. These
17658 parameters may be specified by explicit application execution of the *sched_setscheduler()* or
17659 *sched_setparam()* functions.

17660 The effect of the process scheduling attributes on individual threads in the process is dependent
17661 on the scheduling contention scope of the threads (see [Section 2.9.4](#), on page 515):

17662 For threads with system scheduling contention scope, the process scheduling attributes
17663 shall have no effect on the scheduling attributes or behavior either of the thread or an
17664 underlying kernel scheduling entity dedicated to that thread.

17665 For threads with process scheduling contention scope, the process scheduling attributes
17666 shall have no effect on the scheduling attributes of the thread. However, any underlying
17667 kernel scheduling entity used by these threads shall at all times behave as specified by the
17668 scheduling attributes of the containing process, and this behavior may affect the
17669 scheduling behavior of the process contention scope threads. For example, a process
17670 contention scope thread with scheduling policy SCHED_FIFO and the system maximum
17671 priority *H* (the value returned by *sched_get_priority_max(SCHED_FIFO)*) in a process with
17672 scheduling policy SCHED_RR and system minimum priority *L* (the value returned by
17673 *sched_get_priority_min(SCHED_RR)*) shall be subject to timeslicing and to preemption by
17674 any thread with an effective priority higher than *L*.

17675 Associated with each policy is a priority range. Each policy definition shall specify the minimum
17676 priority range for that policy. The priority ranges for each policy may but need not overlap the
17677 priority ranges of other policies.

17678 A conforming implementation shall select the thread that is defined as being at the head of the
17679 highest priority non-empty thread list to become a running thread, regardless of its associated
17680 policy. This thread is then removed from its thread list.

17681 Four scheduling policies are specifically required. Other implementation-defined scheduling
17682 policies may be defined. The following symbols are defined in the Base Definitions volume of
17683 POSIX.1-2017, **<sched.h>**:

17684 SCHED_FIFO First in, first out (FIFO) scheduling policy.

17685 SCHED_RR Round robin scheduling policy.

17686 SS SCHED_SPORADIC Sporadic server scheduling policy.

17687 SCHED_OTHER Another scheduling policy.

17688 The values of these symbols shall be distinct.

17689 SCHED_FIFO

17690 Conforming implementations shall include a scheduling policy called the FIFO scheduling
17691 policy.

17692 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its
17693 threads have been on the list without being executed; generally, the head of the list is the thread
17694 that has been on the list the longest time, and the tail is the thread that has been on the list the
17695 shortest time.

17696 Under the SCHED_FIFO policy, the modification of the definitional thread lists is as follows:

- 17697 1. When a running thread becomes a preempted thread, it becomes the head of the thread
17698 list for its priority.
- 17699 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list
17700 for its priority.
- 17701 3. When a running thread calls the *sched_setscheduler()* function, the process specified in the
17702 function call is modified to the specified policy and the priority specified by the *param*
17703 argument.
- 17704 4. When a running thread calls the *sched_setparam()* function, the priority of the process
17705 specified in the function call is modified to the priority specified by the *param* argument.
- 17706 5. When a running thread calls the *pthread_setschedparam()* function, the thread specified in
17707 the function call is modified to the specified policy and the priority specified by the *param*
17708 argument.
- 17709 6. When a running thread calls the *pthread_setschedprio()* function, the thread specified in the
17710 function call is modified to the priority specified by the *prio* argument.
- 17711 7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio()*
17712 is a running thread or is runnable, it then becomes the tail of the thread list for its new
17713 priority.
- 17714 8. If a thread whose priority has been modified by *pthread_setschedprio()* is a running thread
17715 or is runnable, the effect on its position in the thread list depends on the direction of the
17716 modification, as follows:
- 17717 a. If the priority is raised, the thread becomes the tail of the thread list.
- 17718 b. If the priority is unchanged, the thread does not change position in the thread list.
- 17719 c. If the priority is lowered, the thread becomes the head of the thread list.
- 17720 9. When a running thread issues the *sched_yield()* function, the thread becomes the tail of
17721 the thread list for its priority.
- 17722 10. At no other time is the position of a thread with this scheduling policy within the thread
17723 lists affected.

17724 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
17725 and *sched_get_priority_min()* functions when SCHED_FIFO is provided as the parameter.
17726 Conforming implementations shall provide a priority range of at least 32 priorities for this
17727 policy.

17728 SCHED_RR

17729 Conforming implementations shall include a scheduling policy called the “round robin”
17730 scheduling policy. This policy shall be identical to the SCHED_FIFO policy with the additional
17731 condition that when the implementation detects that a running thread has been executing as a
17732 running thread for a time period of the length returned by the *sched_rr_get_interval()* function or
17733 longer, the thread shall become the tail of its thread list and the head of that thread list shall be
17734 removed and made a running thread.

17735 The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same
17736 priority, one of them does not monopolize the processor. An application should not rely only on
17737 the use of SCHED_RR to ensure application progress among multiple threads if the application
17738 includes threads using the SCHED_FIFO policy at the same or higher priority levels or
17739 SCHED_RR threads at a higher priority level.

17740 A thread under this policy that is preempted and subsequently resumes execution as a running

17741 thread completes the unexpired portion of its round robin interval time period.

17742 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
 17743 and *sched_get_priority_min()* functions when SCHED_RR is provided as the parameter.
 17744 Conforming implementations shall provide a priority range of at least 32 priorities for this
 17745 policy.

17746 SCHED_SPORADIC

17747 SS|TSP The functionality described in this section shall be provided on implementations that support
 17748 the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not
 17749 further shaded for these options).

17750 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the
 17751 implementation shall include a scheduling policy identified by the value SCHED_SPORADIC.

17752 The sporadic server policy is based primarily on two time parameters: the replenishment period
 17753 and the available execution capacity. The replenishment period is given by the
 17754 *sched_ss_repl_period* member of the **sched_param** structure. The available execution capacity is
 17755 initialized to the value given by the *sched_ss_init_budget* member of the same parameter. The
 17756 sporadic server policy is identical to the SCHED_FIFO policy with some additional conditions
 17757 that cause the thread's assigned priority to be switched between the values specified by the
 17758 *sched_priority* and *sched_ss_low_priority* members of the **sched_param** structure.

17759 The priority assigned to a thread using the sporadic server scheduling policy is determined in
 17760 the following manner: if the available execution capacity is greater than zero and the number of
 17761 pending replenishment operations is strictly less than *sched_ss_max_repl*, the thread is assigned
 17762 the priority specified by *sched_priority*; otherwise, the assigned priority shall be
 17763 *sched_ss_low_priority*. If the value of *sched_priority* is less than or equal to the value of
 17764 *sched_ss_low_priority*, the results are undefined. When active, the thread shall belong to the
 17765 thread list corresponding to its assigned priority level, according to the mentioned priority
 17766 assignment. The modification of the available execution capacity and, consequently of the
 17767 assigned priority, is done as follows:

- 17768 1. When the thread at the head of the *sched_priority* list becomes a running thread, its
 17769 execution time shall be limited to at most its available execution capacity, plus the
 17770 resolution of the execution time clock used for this scheduling policy. This resolution shall
 17771 be implementation-defined.
- 17772 2. Each time the thread is inserted at the tail of the list associated with *sched_priority* †
 17773 because as a blocked thread it became runnable with priority *sched_priority* or because a
 17774 replenishment operation was performed ‡the time at which this operation is done is
 17775 posted as the *activation_time*.
- 17776 3. When the running thread with assigned priority equal to *sched_priority* becomes a
 17777 preempted thread, it becomes the head of the thread list for its priority, and the execution
 17778 time consumed is subtracted from the available execution capacity. If the available
 17779 execution capacity would become negative by this operation, it shall be set to zero.
- 17780 4. When the running thread with assigned priority equal to *sched_priority* becomes a blocked
 17781 thread, the execution time consumed is subtracted from the available execution capacity,
 17782 and a replenishment operation is scheduled, as described in 6 and 7. If the available
 17783 execution capacity would become negative by this operation, it shall be set to zero.
- 17784 5. When the running thread with assigned priority equal to *sched_priority* reaches the limit
 17785 imposed on its execution time, it becomes the tail of the thread list for
 17786 *sched_ss_low_priority*, the execution time consumed is subtracted from the available
 17787 execution capacity (which becomes zero), and a replenishment operation is scheduled, as

- 17788 described in 6 and 7.
- 17789 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be
 17790 replenished, *replenish_amount*, is set equal to the execution time consumed by the thread
 17791 since the *activation_time*. The replenishment is scheduled to occur at *activation_time* plus
 17792 *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the
 17793 replenishment operation is carried out immediately. Several replenishment operations
 17794 may be pending at the same time, each of which will be serviced at its respective
 17795 scheduled time. With the above rules, the number of replenishment operations
 17796 simultaneously pending for a given thread that is scheduled under the sporadic server
 17797 policy shall not be greater than *sched_ss_max_repl*.
- 17798 7. A replenishment operation consists of adding the corresponding *replenish_amount* to the
 17799 available execution capacity at the scheduled time. If, as a consequence of this operation,
 17800 the execution capacity would become larger than *sched_ss_initial_budget*, it shall be
 17801 rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was
 17802 runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it
 17803 becomes the tail of the thread list for *sched_priority*.
- 17804 Execution time is defined in XBD [Section 3.118](#) (on page 52).
- 17805 For this policy, changing the value of a CPU-time clock via *clock_settime()* shall have no effect on
 17806 its behavior.
- 17807 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min()*
 17808 and *sched_get_priority_max()* functions when SCHED_SPORADIC is provided as the parameter.
 17809 Conforming implementations shall provide a priority range of at least 32 distinct priorities for
 17810 this policy.
- 17811 If the scheduling policy of the target process is either SCHED_FIFO or SCHED_RR, the
 17812 *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init* budget members of the *param*
 17813 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process
 17814 is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the effects of these members are
 17815 implementation-defined; this case includes the SCHED_OTHER policy.
- 17816 **SCHED_OTHER**
- 17817 Conforming implementations shall include one scheduling policy identified as SCHED_OTHER
 17818 (which may execute identically with either the FIFO or round robin scheduling policy). The
 17819 effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads
 17820 SS are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-
 17821 defined.
- 17822 This policy is defined to allow strictly conforming applications to be able to indicate in a
 17823 portable manner that they no longer need a realtime scheduling policy.
- 17824 For threads executing under this policy, the implementation shall use only priorities within the
 17825 range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when
 17826 SCHED_OTHER is provided as the parameter.

17827 **2.8.5 Clocks and Timers**

17828 The `<time.h>` header defines the types and manifest constants used by the timing facility.

17829 **Time Value Specification Structures**

17830 Many of the timing facility functions accept or return time value specifications. A time value
17831 structure `timespec` specifies a single time value and includes at least the following members:

Member Type	Member Name	Description
<code>time_t</code>	<code>tv_sec</code>	Seconds.
<code>long</code>	<code>tv_nsec</code>	Nanoseconds.

17835 The `tv_nsec` member is only valid if greater than or equal to zero, and less than the number of
17836 nanoseconds in a second (1 000 million). The time interval described by this structure is $(tv_sec * 10^9 + tv_nsec)$ nanoseconds.

17838 A time value structure `itimerspec` specifies an initial timer value and a repetition interval for use
17839 by the per-process timer functions. This structure includes at least the following members:

Member Type	Member Name	Description
<code>struct timespec</code>	<code>it_interval</code>	Timer period.
<code>struct timespec</code>	<code>it_value</code>	Timer expiration.

17843 If the value described by `it_value` is non-zero, it indicates the time to or time of the next timer
17844 expiration (for relative and absolute timer values, respectively). If the value described by `it_value`
17845 is zero, the timer shall be disarmed.

17846 If the value described by `it_interval` is non-zero, it specifies an interval which shall be used in
17847 reloading the timer when it expires; that is, a periodic timer is specified. If the value described
17848 by `it_interval` is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is
17849 specified.

17850 **Timer Event Notification Control Block**

17851 Per-process timers may be created that notify the process of timer expirations by queuing a
17852 realtime extended signal. The `sigevent` structure, defined in the Base Definitions volume of
17853 POSIX.1-2017, `<signal.h>`, is used in creating such a timer. The `sigevent` structure contains the
17854 signal number and an application-specific data value which shall be used when notifying the
17855 calling process of timer expiration events.

17856 **Manifest Constants**

17857 The following constants are defined in the Base Definitions volume of POSIX.1-2017, `<time.h>`:

17858	<code>CLOCK_REALTIME</code>	The identifier for the system-wide realtime clock.
17859	<code>TIMER_ABSTIME</code>	Flag indicating time is absolute with respect to the clock associated with a timer.
17861	<code>CLOCK_MONOTONIC</code>	The identifier for the system-wide monotonic clock, which is defined as a clock whose value cannot be set via <code>clock_settime()</code> and which cannot have backward clock jumps. The maximum possible clock jump is implementation-defined.
17862		
17863		
17864		
17865	<code>MON</code>	The maximum allowable resolution for <code>CLOCK_REALTIME</code> and <code>CLOCK_MONOTONIC</code> clocks and all time services based on these clocks is represented by <code>{_POSIX_CLOCKRES_MIN}</code> and shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of
17866		
17867		

17868 resolution for these clocks to provide finer granularity time bases. The actual resolution
 17869 supported by an implementation for a specific clock is obtained using the `clock_getres()`
 17870 function. If the actual resolution supported for a time service based on one of these clocks differs from the
 17871 resolution supported for that clock, the implementation shall document this difference.

17872 MON The minimum allowable maximum value for `CLOCK_REALTIME` and `CLOCK_MONOTONIC`
 17873 clocks and all absolute time services based on them is the same as that defined by the ISO C
 17874 standard for the `time_t` type. If the maximum value supported by a time service based on one of
 17875 these clocks differs from the maximum value supported by that clock, the implementation shall
 17876 document this difference.

17877 Execution Time Monitoring

17878 CPT If `_POSIX_CPUTIME` is defined, process CPU-time clocks shall be supported in addition to the
 17879 clocks described in [Manifest Constants](#) (on page 511).

17880 TCT If `_POSIX_THREAD_CPUTIME` is defined, thread CPU-time clocks shall be supported.

17881 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in XBD [Section 3.118](#) (on
 17882 page 52). The mechanism used to measure execution time is described in XBD [Section 4.11](#) (on
 17883 page 110).

17884 CPT If `_POSIX_CPUTIME` is defined, the following constant of the type `clockid_t` is defined in
 17885 `<time.h>`:

17886 `CLOCK_PROCESS_CPUTIME_ID`

17887 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is
 17888 interpreted as the identifier of the CPU-time clock associated with the process making the
 17889 function call.

17890 TCT If `_POSIX_THREAD_CPUTIME` is defined, the following constant of the type `clockid_t` is
 17891 defined in `<time.h>`:

17892 `CLOCK_THREAD_CPUTIME_ID`

17893 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is
 17894 interpreted as the identifier of the CPU-time clock associated with the thread making the
 17895 function call.

17896 2.9 Threads

17897 This section defines functionality to support multiple flows of control, called “threads”, within a
 17898 process. For the definition of threads, see XBD [Section 3.404](#) (on page 99).

17899 The specific functional areas covered by threads and their scope include:

17900 Thread management: the creation, control, and termination of multiple flows of control in
 17901 the same process under the assumption of a common shared address space

17902 Synchronization primitives optimized for tightly coupled operation of multiple control
 17903 flows in a common, shared address space

17904 **2.9.1 Thread-Safety**

17905 All functions defined by this volume of POSIX.1-2017 shall be thread-safe, except that the
17906 following functions⁷ need not be thread-safe.

17907	<i>asctime()</i>	<i>ftw()</i>	<i>getutxent()</i>	<i>putenv()</i>
17908	<i>basename()</i>	<i>getdate()</i>	<i>getutxid()</i>	<i>pututxline()</i>
17909	<i>catgets()</i>	<i>getenv()</i>	<i>getutxline()</i>	<i>rand()</i>
17910	<i>crypt()</i>	<i>getgrent()</i>	<i>gmtime()</i>	<i>readdir()</i>
17911	<i>ctime()</i>	<i>getgrgid()</i>	<i>hcreate()</i>	<i>setenv()</i>
17912	<i>dbm_clearerr()</i>	<i>getgrnam()</i>	<i>hdestroy()</i>	<i>setgrent()</i>
17913	<i>dbm_close()</i>	<i>gethostent()</i>	<i>hsearch()</i>	<i>setkey()</i>
17914	<i>dbm_delete()</i>	<i>getlogin()</i>	<i>inet_ntoa()</i>	<i>setlocale()</i>
17915	<i>dbm_error()</i>	<i>getnetbyaddr()</i>	<i>l64a()</i>	<i>setpwent()</i>
17916	<i>dbm_fetch()</i>	<i>getnetbyname()</i>	<i>lgamma()</i>	<i>setutxent()</i>
17917	<i>dbm_firstkey()</i>	<i>getnetent()</i>	<i>lgammaf()</i>	<i>strerror()</i>
17918	<i>dbm_nextkey()</i>	<i>getopt()</i>	<i>lgammal()</i>	<i>strsignal()</i>
17919	<i>dbm_open()</i>	<i>getprotobyname()</i>	<i>localeconv()</i>	<i>strtok()</i>
17920	<i>dbm_store()</i>	<i>getprotobynumber()</i>	<i>localtime()</i>	<i>system()</i>
17921	<i>dirname()</i>	<i>getprotoent()</i>	<i>lrand48()</i>	<i>ttyname()</i>
17922	<i>dllerror()</i>	<i>getpwent()</i>	<i>mblen()</i>	<i>unsetenv()</i>
17923	<i>drand48()</i>	<i>getpwnam()</i>	<i>mbtowc()</i>	<i>wctomb()</i>
17924	<i>encrypt()</i>	<i>getpwuid()</i>	<i>mrnd48()</i>	
17925	<i>endgrent()</i>	<i>getserobyname()</i>	<i>nftw()</i>	
17926	<i>endpwent()</i>	<i>getserobyport()</i>	<i>nl_langinfo()</i>	
17927	<i>endutxent()</i>	<i>getservent()</i>	<i>ptsname()</i>	

17928 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The
17929 *mbrlen()*, *mbrtowc()*, *mbsnrto wcs()*, *mbsrtowcs()*, *wcrtomb()*, *wcsnrto mbs()*, and *wcsrtombs()*
17930 functions need not be thread-safe if passed a NULL *ps* argument. The *getc_unlocked()*,
17931 *getchar_unlocked()*, *putc_unlocked()*, and *putchar_unlocked()* functions need not be thread-safe
17932 unless the invoking thread owns the (FILE *) object accessed by the call, as is the case after a
17933 successful call to the *flockfile()* or *ftrylockfile()* functions.

17934 Implementations shall provide internal synchronization as necessary in order to satisfy this
17935 requirement.

17936 Since multi-threaded applications are not allowed to use the *environ* variable to access or modify
17937 any environment variable while any other thread is concurrently modifying any environment
17938 variable, any function dependent on any environment variable is not thread-safe if another
17939 thread is modifying the environment; see XSH *exec* (on page 783).

17940 **2.9.2 Thread IDs**

17941 Although implementations may have thread IDs that are unique in a system, applications
17942 should only assume that thread IDs are usable and unique within a single process. The effect of
17943 calling any of the functions defined in this volume of POSIX.1-2017 and passing as an argument
17944 the thread ID of a thread from another process is unspecified. The lifetime of a thread ID ends
17945 after the thread terminates if it was created with the *detachstate* attribute set to
17946 PTHREAD_CREATE_DETACHED or if *pthread_detach()* or *pthread_join()* has been called for that
17947 thread. A conforming implementation is free to reuse a thread ID after its lifetime has ended. If
17948 an application attempts to use a thread ID whose lifetime has ended, the behavior is undefined.

17949 7. The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

17950 If a thread is detached, its thread ID is invalid for use as an argument in a call to *pthread_detach()*
17951 or *pthread_join()*.

17952 2.9.3 Thread Mutexes

17953 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same
17954 processing resources from eventually making forward progress in its execution. Eligibility for
17955 processing resources is determined by the scheduling policy.

17956 A thread shall become the owner of a mutex, *m*, when one of the following occurs:

17957 It calls *pthread_mutex_lock()* with *m* as the *mutex* argument and the call returns zero or
17958 [EOWNERDEAD].

17959 It calls *pthread_mutex_trylock()* with *m* as the *mutex* argument and the call returns zero or
17960 [EOWNERDEAD].

17961 It calls *pthread_mutex_timedlock()* with *m* as the *mutex* argument and the call returns zero or
17962 [EOWNERDEAD].

17963 It calls *pthread_mutex_setprioceiling()* with *m* as the *mutex* argument and the call returns
17964 [EOWNERDEAD].

17965 It calls *pthread_cond_wait()* with *m* as the *mutex* argument and the call returns zero or
17966 certain error numbers (see *pthread_cond_timedwait()*).

17967 It calls *pthread_cond_timedwait()* with *m* as the *mutex* argument and the call returns zero or
17968 certain error numbers (see *pthread_cond_timedwait()*).

17969 The thread shall remain the owner of *m* until one of the following occurs:

17970 It executes *pthread_mutex_unlock()* with *m* as the *mutex* argument

17971 It blocks in a call to *pthread_cond_wait()* with *m* as the *mutex* argument.

17972 It blocks in a call to *pthread_cond_timedwait()* with *m* as the *mutex* argument.

17973 The implementation shall behave as if at all times there is at most one owner of any mutex.

17974 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex
17975 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have
17976 “released” the mutex and the mutex is said to have become “unlocked”.

17977 A problem can occur if a process terminates while one of its threads holds a mutex lock.
17978 Depending on the mutex type, it might be possible for another thread to unlock the mutex and
17979 recover the state of the mutex. However, it is difficult to perform this recovery reliably.

17980 Robust mutexes provide a means to enable the implementation to notify other threads in the
17981 event of a process terminating while one of its threads holds a mutex lock. The next thread that
17982 acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from
17983 the locking function. The notified thread can then attempt to recover the state protected by the
17984 mutex, and if successful mark the state protected by the mutex as consistent by a call to
17985 *pthread_mutex_consistent()*. If the notified thread is unable to recover the state, it can declare the
17986 state as not recoverable by a call to *pthread_mutex_unlock()* without a prior call to
17987 *pthread_mutex_consistent()*.

17988 Whether or not the state protected by a mutex can be recovered is dependent solely on the
17989 application using robust mutexes. The robust mutex support provided in the implementation
17990 provides notification only that a mutex owner has terminated while holding a lock, or that the
17991 state of the mutex is not recoverable.

17992 **2.9.4 Thread Scheduling**

17993 TPS The functionality described in this section shall be provided on implementations that support
 17994 the Thread Execution Scheduling option (and the rest of this section is not further shaded for
 17995 this option).

17996 **Thread Scheduling Attributes**

17997 In support of the scheduling function, threads have attributes which are accessed through the
 17998 **pthread_attr_t** thread creation attributes object.

17999 The *contentionscope* attribute defines the scheduling contention scope of the thread to be either
 18000 PTHREAD_SCOPE_PROCESS or PTHREAD_SCOPE_SYSTEM.

18001 The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling
 18002 attributes of the creating thread or to have its scheduling values set according to the other
 18003 scheduling attributes in the **pthread_attr_t** object.

18004 The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute
 18005 defines the scheduling parameters for the thread. The interaction of threads having different
 18006 policies within a process is described as part of the definition of those policies.

18007 If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one
 18008 of the priority-based policies defined under this option, the *schedparam* attribute contains the
 18009 scheduling priority of the thread. A conforming implementation ensures that the priority value
 18010 in *schedparam* is in the range associated with the scheduling policy when the thread attributes
 18011 object is used to create a thread, or when the scheduling attributes of a thread are dynamically
 18012 modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

18013 TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four
 18014 new members that are used for the sporadic server scheduling policy. These members are
 18015 *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The
 18016 meaning of these attributes is the same as in the definitions that appear under [Section 2.8.4](#) (on
 18017 page 506).

18018 When a process is created, its single thread has a scheduling policy and associated attributes
 18019 equal to the policy and attributes of the process. The default scheduling contention scope value
 18020 is implementation-defined. The default values of other scheduling attributes are
 18021 implementation-defined.

18022 **Thread Scheduling Contention Scope**

18023 The scheduling contention scope of a thread defines the set of threads with which the thread
 18024 competes for use of the processing resources. The scheduling operation selects at most one
 18025 thread to execute on each processor at any point in time and the thread's scheduling attributes
 18026 (for example, *priority*), whether under process scheduling contention scope or system scheduling
 18027 contention scope, are the parameters used to determine the scheduling decision.

18028 The scheduling contention scope, in the context of scheduling a mixed scope environment,
 18029 affects threads as follows:

18030 A thread created with PTHREAD_SCOPE_SYSTEM scheduling contention scope contends
 18031 for resources with all other threads in the same scheduling allocation domain relative to
 18032 their system scheduling attributes. The system scheduling attributes of a thread created
 18033 with PTHREAD_SCOPE_SYSTEM scheduling contention scope are the scheduling
 18034 attributes with which the thread was created. The system scheduling attributes of a thread
 18035 created with PTHREAD_SCOPE_PROCESS scheduling contention scope are the
 18036 implementation-defined mapping into system attribute space of the scheduling attributes

18037 with which the thread was created.

18038 Threads created with PTHREAD_SCOPE_PROCESS scheduling contention scope contend
18039 directly with other threads within their process that were created with
18040 PTHREAD_SCOPE_PROCESS scheduling contention scope. The contention is resolved
18041 based on the threads' scheduling attributes and policies. It is unspecified how such threads
18042 are scheduled relative to threads in other processes or threads with
18043 PTHREAD_SCOPE_SYSTEM scheduling contention scope.

18044 Conforming implementations shall support the PTHREAD_SCOPE_PROCESS scheduling
18045 contention scope, the PTHREAD_SCOPE_SYSTEM scheduling contention scope, or both.

18046 **Scheduling Allocation Domain**

18047 Implementations shall support scheduling allocation domains containing one or more
18048 processors. It should be noted that the presence of multiple processors does not automatically
18049 indicate a scheduling allocation domain size greater than one. Conforming implementations on
18050 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation
18051 domains, and could define these scheduling allocation domains on a per-thread, per-process, or
18052 per-system basis, depending on the types of applications intended to be supported by the
18053 implementation. The scheduling allocation domain is independent of scheduling contention
18054 scope, as the scheduling contention scope merely defines the set of threads with which a thread
18055 contends for processor resources, while scheduling allocation domain defines the set of
18056 processors for which it contends. The semantics of how this contention is resolved among
18057 threads for processors is determined by the scheduling policies of the threads.

18058 The choice of scheduling allocation domain size and the level of application control over
18059 scheduling allocation domains is implementation-defined. Conforming implementations may
18060 change the size of scheduling allocation domains and the binding of threads to scheduling
18061 allocation domains at any time.

18062 For application threads with scheduling allocation domains of size equal to one, the scheduling
18063 rules defined for SCHED_FIFO and SCHED_RR shall be used; see [Scheduling Policies](#) (on page
18064 506). All threads with system scheduling contention scope, regardless of the processes in which
18065 they reside, compete for the processor according to their priorities. Threads with process
18066 scheduling contention scope compete only with other threads with process scheduling
18067 contention scope within their process.

18068 For application threads with scheduling allocation domains of size greater than one, the rules
18069 TSP defined for SCHED_FIFO, SCHED_RR, and SCHED_SPORADIC shall be used in an
18070 implementation-defined manner. Each thread with system scheduling contention scope
18071 competes for the processors in its scheduling allocation domain in an implementation-defined
18072 manner according to its priority. Threads with process scheduling contention scope are
18073 scheduled relative to other threads within the same scheduling contention scope in the process.

18074 TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, the rules defined for SCHED_SPORADIC
18075 in [Scheduling Policies](#) (on page 506) shall be used in an implementation-defined manner for
18076 application threads whose scheduling allocation domain size is greater than one.

18077 **Scheduling Documentation**

18078 If `_POSIX_PRIORITY_SCHEDULING` is defined, then any scheduling policies beyond
 18079 TSP `SCHED_OTHER`, `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC`, as well as the effects of
 18080 the scheduling policies indicated by these other values, and the attributes required in order to
 18081 support such a policy, are implementation-defined. Furthermore, the implementation shall
 18082 document the effect of all processor scheduling allocation domain values supported for these
 18083 policies.

18084 **2.9.5 Thread Cancellation**

18085 The thread cancellation mechanism allows a thread to terminate the execution of any other
 18086 thread in the process in a controlled manner. The target thread (that is, the one that is being
 18087 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform
 18088 application-specific cleanup processing when the notice of cancellation is acted upon.

18089 Cancellation is controlled by the cancellation control functions. Each thread maintains its own
 18090 cancelability state. Cancellation may only occur at cancellation points or when the thread is
 18091 asynchronously cancelable.

18092 The thread cancellation mechanism described in this section depends upon programs having set
 18093 *deferred* cancelability state, which is specified as the default. Applications shall also carefully
 18094 follow static lexical scoping rules in their execution behavior. For example, use of `setjmp()`,
 18095 `return`, `goto`, and so on, to leave user-defined cancellation scopes without doing the necessary
 18096 scope pop operation results in undefined behavior.

18097 Use of asynchronous cancelability while holding resources which potentially need to be released
 18098 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated
 18099 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

18100 **2.9.5.1 Cancelability States**

18101 The cancelability state of a thread determines the action taken upon receipt of a cancellation
 18102 request. The thread may control cancellation in a number of ways.

18103 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 18104 1. Cancelability-Enable: When cancelability is `PTHREAD_CANCEL_DISABLE` (as defined
 18105 in the Base Definitions volume of POSIX.1-2017, `<pthread.h>`), cancellation requests
 18106 against the target thread are held pending. By default, cancelability is set to
 18107 `PTHREAD_CANCEL_ENABLE` (as defined in `<pthread.h>`).
- 18108 2. Cancelability Type: When cancelability is enabled and the cancelability type is
 18109 `PTHREAD_CANCEL_ASYNCHRONOUS` (as defined in `<pthread.h>`), new or pending
 18110 cancellation requests may be acted upon at any time. When cancelability is enabled and
 18111 the cancelability type is `PTHREAD_CANCEL_DEFERRED` (as defined in `<pthread.h>`),
 18112 cancellation requests are held pending until a cancellation point (see below) is reached. If
 18113 cancelability is disabled, the setting of the cancelability type has no immediate effect as all
 18114 cancellation requests are held pending; however, once cancelability is enabled again the
 18115 new type is in effect. The cancelability type is `PTHREAD_CANCEL_DEFERRED` in all
 18116 newly created threads including the thread in which `main()` was first invoked.

18117 2.9.5.2 Cancellation Points

18118 Cancellation points shall occur when a thread is executing the following functions:

18119	<i>accept()</i>	<i>nanosleep()</i>	<i>select()</i>
18120	<i>aiow_suspend()</i>	<i>open()</i>	<i>sem_timedwait()</i>
18121	<i>clock_nanosleep()</i>	<i>openat()</i>	<i>sem_wait()</i>
18122	<i>close()</i>	<i>pause()</i>	<i>send()</i>
18123	<i>connect()</i>	<i>poll()</i>	<i>sendmsg()</i>
18124	<i>creat()</i>	<i>pread()</i>	<i>sendto()</i>
18125	<i>fcntl()</i> •	<i>pselect()</i>	<i>sigsuspend()</i>
18126	<i>fdatasync()</i>	<i>pthread_cond_timedwait()</i>	<i>sigtimedwait()</i>
18127	<i>fsync()</i>	<i>pthread_cond_wait()</i>	<i>sigwait()</i>
18128	<i>getmsg()</i>	<i>pthread_join()</i>	<i>sigwaitinfo()</i>
18129	<i>getpmsg()</i>	<i>pthread_testcancel()</i>	<i>sleep()</i>
18130	<i>lockf()</i> • •	<i>putmsg()</i>	<i>tcdrain()</i>
18131	<i>mq_receive()</i>	<i>putpmsg()</i>	<i>wait()</i>
18132	<i>mq_send()</i>	<i>pwrite()</i>	<i>waitid()</i>
18133	<i>mq_timedreceive()</i>	<i>read()</i>	<i>waitpid()</i>
18134	<i>mq_timedsend()</i>	<i>readv()</i>	<i>write()</i>
18135	<i>msgrcv()</i>	<i>recv()</i>	<i>writew()</i>
18136	<i>msgsnd()</i>	<i>recvfrom()</i>	
18137	<i>msync()</i>	<i>recvmsg()</i>	

18138 † When the *cmd* argument is F_SETLKW.

18139 †† When the *function* argument is F_LOCK.

18140 A cancellation point may also occur when a thread is executing the following functions:

18141	<i>access()</i>	<i>fstatat()</i>	<i>mkstemp()</i>
18142	<i>asctime_r()</i>	<i>ftell()</i>	<i>mktime()</i>
18143	<i>catclose()</i>	<i>ftello()</i>	<i>opendir()</i>
18144	<i>catopen()</i>	<i>futimens()</i>	<i>openlog()</i>
18145	<i>chmod()</i>	<i>fwprintf()</i>	<i>pathconf()</i>
18146	<i>chown()</i>	<i>fwrite()</i>	<i>perror()</i>
18147	<i>closedir()</i>	<i>fwscanf()</i>	<i>popen()</i>
18148	<i>closelog()</i>	<i>getaddrinfo()</i>	<i>posix_fadvise()</i>
18149	<i>ctermid()</i>	<i>getc()</i>	<i>posix_fallocate()</i>
18150	<i>ctime_r()</i>	<i>getc_unlocked()</i>	<i>posix_madvise()</i>
18151	<i>dlclose()</i>	<i>getchar()</i>	<i>posix_openpt()</i>
18152	<i>dlopen()</i>	<i>getchar_unlocked()</i>	<i>posix_spawn()</i>
18153	<i>dprintf()</i>	<i>getcwd()</i>	<i>posix_spawnnp()</i>
18154	<i>endhostent()</i>	<i>getdelim()</i>	<i>posix_trace_clear()</i>
18155	<i>endnetent()</i>	<i>getgrgid_r()</i>	<i>posix_trace_close()</i>
18156	<i>endprotoent()</i>	<i>getgrnam_r()</i>	<i>posix_trace_create()</i>
18157	<i>endservent()</i>	<i>gethostid()</i>	<i>posix_trace_create_withlog()</i>
18158	<i>faccessat()</i>	<i>gethostname()</i>	<i>posix_trace_eventtypelist_getnext_id()</i>
18159	<i>fchmod()</i>	<i>getline()</i>	<i>posix_trace_eventtypelist_rewind()</i>
18160	<i>fchmodat()</i>	<i>getlogin_r()</i>	<i>posix_trace_flush()</i>
18161	<i>fchown()</i>	<i>getnameinfo()</i>	<i>posix_trace_get_attr()</i>
18162	<i>fchownat()</i>	<i>getpwnam_r()</i>	<i>posix_trace_get_filter()</i>
18163	<i>fclose()</i>	<i>getpwuid_r()</i>	<i>posix_trace_get_status()</i>
18164	<i>fcntl()</i> •	<i>gets()</i>	<i>posix_trace_getnext_event()</i>
18165	<i>fflush()</i>	<i>getwc()</i>	<i>posix_trace_open()</i>
18166	<i>fgetc()</i>	<i>getwchar()</i>	<i>posix_trace_rewind()</i>
18167	<i>fgetpos()</i>	<i>glob()</i>	<i>posix_trace_set_filter()</i>
18168	<i>fgets()</i>	<i>iconv_close()</i>	<i>posix_trace_shutdown()</i>
18169	<i>fgetwc()</i>	<i>iconv_open()</i>	<i>posix_trace_timedgetnext_event()</i>
18170	<i>fgetws()</i>	<i>ioctl()</i>	<i>posix_typed_mem_open()</i>
18171	<i>fntmsg()</i>	<i>link()</i>	<i>printf()</i>
18172	<i>fopen()</i>	<i>linkat()</i>	<i>psiginfo()</i>
18173	<i>fpathconf()</i>	<i>lio_listio()</i>	<i>psignal()</i>
18174	<i>fprintf()</i>	<i>localtime_r()</i>	<i>pthread_rwlock_rdlock()</i>
18175	<i>fputc()</i>	<i>lockf()</i>	<i>pthread_rwlock_timedrdlock()</i>
18176	<i>fputs()</i>	<i>lseek()</i>	<i>pthread_rwlock_timedwrlock()</i>
18177	<i>fputwc()</i>	<i>lstat()</i>	<i>pthread_rwlock_wrlock()</i>
18178	<i>fputws()</i>	<i>mkdir()</i>	<i>putc()</i>
18179	<i>fread()</i>	<i>mkdirat()</i>	<i>putc_unlocked()</i>
18180	<i>freopen()</i>	<i>mkdtemp()</i>	<i>putchar()</i>
18181	<i>fscanf()</i>	<i>mkfifo()</i>	<i>putchar_unlocked()</i>
18182	<i>fseek()</i>	<i>mkfifoat()</i>	<i>puts()</i>
18183	<i>fseeko()</i>	<i>mknod()</i>	<i>putwc()</i>
18184	<i>fsetpos()</i>	<i>mknodat()</i>	<i>putwchar()</i>
18185	<i>fstat()</i>		<i>readdir_r()</i>

18186	<i>readlink()</i>	<i>sigpause()</i>	<i>ungetwc()</i>
18187	<i>readlinkat()</i>	<i>stat()</i>	<i>unlink()</i>
18188	<i>remove()</i>	<i>strerror_l()</i>	<i>unlinkat()</i>
18189	<i>rename()</i>	<i>strerror_r()</i>	<i>utime()</i>
18190	<i>renameat()</i>	<i>strftime()</i>	<i>utimensat()</i>
18191	<i>rewind()</i>	<i>strtime_l()</i>	<i>utimes()</i>
18192	<i>rewinddir()</i>	<i>symlink()</i>	<i>vdprintf()</i>
18193	<i>scandir()</i>	<i>symlinkat()</i>	<i>vfprintf()</i>
18194	<i>scanf()</i>	<i>sync()</i>	<i>vfwprintf()</i>
18195	<i>seekdir()</i>	<i>syslog()</i>	<i>vprintf()</i>
18196	<i>semop()</i>	<i>tmpfile()</i>	<i>vwprintf()</i>
18197	<i>sethostent()</i>	<i>tmpnam()</i>	<i>wcsftime()</i>
18198	<i>setnetent()</i>	<i>ttyname_r()</i>	<i>wordexp()</i>
18199	<i>setprotoent()</i>	<i>tzset()</i>	<i>wprintf()</i>
18200	<i>setservent()</i>	<i>ungetc()</i>	<i>wscanf()</i>

18201 In addition, a cancellation point may occur when a thread is executing any function that this
 18202 standard does not require to be thread-safe but the implementation documents as being thread-
 18203 safe. If a thread is cancelled while executing a non-thread-safe function, the behavior is
 18204 undefined.

18205 An implementation shall not introduce cancellation points into any other functions specified in
 18206 this volume of POSIX.1-2017.

18207 The side-effects of acting upon a cancellation request while suspended during a call of a function
 18208 are the same as the side-effects that may be seen in a single-threaded program when a call to a
 18209 function is interrupted by a signal and the given function returns [EINTR]. Any such side-
 18210 effects occur before any cancellation cleanup handlers are called. For functions that are explicitly
 18211 required not to return when interrupted (for example, *pclose()*), if a thread is canceled while
 18212 executing the function, the behavior is undefined.

18213 Whenever a thread has cancelability enabled and a cancellation request has been made with that
 18214 thread as the target, and the thread then calls any function that is a cancellation point (such as
 18215 *pthread_testcancel()* or *read()*), the cancellation request shall be acted upon before the function
 18216 returns. If a thread has cancelability enabled and a cancellation request is made with the thread
 18217 as a target while the thread is suspended at a cancellation point, the thread shall be awakened
 18218 and the cancellation request shall be acted upon. It is unspecified whether the cancellation
 18219 request is acted upon or whether the cancellation request remains pending and the thread
 18220 resumes normal execution if:

18221 The thread is suspended at a cancellation point and the event for which it is waiting occurs

18222 A specified timeout expired

18223 before the cancellation request is acted upon.

18224 2.9.5.3 Thread Cancellation Cleanup Handlers

18225 Each thread maintains a list of cancellation cleanup handlers. The programmer uses the
 18226 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions to place routines on and remove
 18227 routines from this list.

18228 When a cancellation request is acted upon, or when a thread calls *pthread_exit()*, the thread first
 18229 disables cancellation by setting its cancelability state to `PTHREAD_CANCEL_DISABLE` and its

18230 For any value of the *cmd* argument.

18231 cancelability type to `PTHREAD_CANCEL_DEFERRED`. The cancelability state shall remain set
 18232 to `PTHREAD_CANCEL_DISABLE` until the thread has terminated. The behavior is undefined if
 18233 a cancellation cleanup handler or thread-specific data destructor routine changes the
 18234 cancelability state to `PTHREAD_CANCEL_ENABLE`.

18235 The routines in the thread's list of cancellation cleanup handlers are invoked one by one in LIFO
 18236 sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First
 18237 Out). When the cancellation cleanup handler for a scope is invoked, the storage for that scope
 18238 remains valid. If the last cancellation cleanup handler returns, thread-specific data destructors (if
 18239 any) associated with thread-specific data keys for which the thread has non-NULL values will
 18240 be run, in unspecified order, as described for *pthread_key_create()*.

18241 After all cancellation cleanup handlers and thread-specific data destructors have returned,
 18242 thread execution is terminated. If the thread has terminated because of a call to *pthread_exit()*,
 18243 the *value_ptr* argument is made available to any threads joining with the target. If the thread has
 18244 terminated by acting on a cancellation request, a status of `PTHREAD_CANCELED` is made
 18245 available to any threads joining with the target. The symbolic constant `PTHREAD_CANCELED`
 18246 expands to a constant expression of type `(void *)` whose value matches no pointer to an object in
 18247 memory nor the value `NULL`.

18248 A side-effect of acting upon a cancellation request while in a condition variable wait is that the
 18249 mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread
 18250 is no longer considered to be waiting for the condition and the thread shall not have consumed
 18251 any pending condition signals on the condition.

18252 A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

18253 2.9.5.4 Async-Cancel Safety

18254 The *pthread_cancel()*, *pthread_setcancelstate()*, and *pthread_setcanceltype()* functions are defined to
 18255 be async-cancel safe.

18256 No other functions in this volume of POSIX.1-2017 are required to be async-cancel-safe.

18257 If a thread has asynchronous cancellation enabled and is cancelled during execution of a
 18258 function that is not async-cancel-safe, the behavior is undefined.

18259 If a thread has deferred cancellation enabled, a signal-catching function is called in that thread
 18260 during execution of a function that is not async-cancel-safe, and the signal-catching function
 18261 calls any function that is a cancellation point while a cancellation is pending for the thread, the
 18262 behavior is undefined.

18263 2.9.6 Thread Read-Write Locks

18264 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
 18265 read-only access to data while allowing only one thread to have exclusive write access at any
 18266 given time. They are typically used to protect data that is read more frequently than it is
 18267 changed.

18268 One or more readers acquire read access to the resource by performing a read lock operation on
 18269 the associated read-write lock. A writer acquires exclusive write access by performing a write
 18270 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and
 18271 any other writers.

18272 A thread that has blocked on a read-write lock (for example, has not yet returned from a
 18273 *pthread_rwlock_rdlock()* or *pthread_rwlock_wrlock()* call) shall not prevent any unblocked thread

18274 that is eligible to use the same processing resources from eventually making forward progress in
 18275 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

18276 Read-write locks can be used to synchronize threads in the current process and other processes if
 18277 they are allocated in memory that is writable and shared among the cooperating processes and
 18278 have been initialized for this behavior.

18279 2.9.7 Thread Interactions with Regular File Operations

18280 All of the following functions shall be atomic with respect to each other in the effects specified in
 18281 POSIX.1-2017 when they operate on regular files or symbolic links:

18282	<i>chmod()</i>	<i>fchownat()</i>	<i>lseek()</i>	<i>readv()</i>	<i>unlink()</i>
18283	<i>chown()</i>	<i>fcntl()</i>	<i>lstat()</i>	<i>pwrite()</i>	<i>unlinkat()</i>
18284	<i>close()</i>	<i>fstat()</i>	<i>open()</i>	<i>rename()</i>	<i>utime()</i>
18285	<i>creat()</i>	<i>fstatat()</i>	<i>openat()</i>	<i>renameat()</i>	<i>utimensat()</i>
18286	<i>dup2()</i>	<i>ftruncate()</i>	<i>pread()</i>	<i>stat()</i>	<i>utimes()</i>
18287	<i>fchmod()</i>	<i>lchown()</i>	<i>read()</i>	<i>symlink()</i>	<i>write()</i>
18288	<i>fchmodat()</i>	<i>link()</i>	<i>readlink()</i>	<i>symlinkat()</i>	<i>writev()</i>
18289	<i>fchown()</i>	<i>linkat()</i>	<i>readlinkat()</i>	<i>truncate()</i>	

18290 If two threads each call one of these functions, each call shall either see all of the specified effects
 18291 of the other call, or none of them. The requirement on the *close()* function shall also apply
 18292 whenever a file descriptor is successfully closed, however caused (for example, as a consequence
 18293 of calling *close()*, calling *dup2()*, or of process termination).

18294 2.9.8 Use of Application-Managed Thread Stacks

18295 An “application-managed thread stack” is a region of memory allocated by the application ¶for
 18296 example, memory returned by the *malloc()* or *mmap()* functions ¶and designated as a stack
 18297 through the act of passing the address and size of the stack, respectively, as the *stackaddr* and
 18298 *stacksize* arguments to *pthread_attr_setstack()*. Application-managed stacks allow the application
 18299 to precisely control the placement and size of a stack.

18300 The application grants to the implementation permanent ownership of and control over the
 18301 application-managed stack when the attributes object in which the *stack* or *stackaddr* attribute has
 18302 been set is used, either by presenting that attribute’s object as the *attr* argument in a call to
 18303 *pthread_create()* that completes successfully, or by storing a pointer to the attributes object in the
 18304 *sigev_notify_attributes* member of a **struct sigevent** and passing that **struct sigevent** to a function
 18305 accepting such argument that completes successfully. The application may thereafter utilize the
 18306 memory within the stack only within the normal context of stack usage within or properly
 18307 synchronized with a thread that has been scheduled by the implementation with stack pointer
 18308 value(s) that are within the range of that stack. In particular, the region of memory cannot be
 18309 freed, nor can it be later specified as the stack for another thread.

18310 When specifying an attributes object with an application-managed stack through the
 18311 *sigev_notify_attributes* member of a **struct sigevent**, the results are undefined if the requested
 18312 signal is generated multiple times (as for a repeating timer).

18313 Until an attributes object in which the *stack* or *stackaddr* attribute has been set is used, the
 18314 application retains ownership of and control over the memory allocated to the stack. It may free
 18315 or reuse the memory as long as it either deletes the attributes object, or before using the
 18316 attributes object replaces the stack by making an additional call to *pthread_attr_setstack()*, that
 18317 was used originally to designate the stack. There is no mechanism to retract the reference to an

18318 application-managed stack by an existing attributes object.

18319 Once an attributes object with an application-managed stack has been used, that attributes object
 18320 cannot be used again by a subsequent call to `pthread_create()` or any function accepting a **struct**
 18321 **sigevent** with `sigev_notify_attributes` containing a pointer to the attributes object, without
 18322 designating an unused application-managed stack by making an additional call to
 18323 `pthread_attr_setstack()`.

18324 2.9.9 Synchronization Object Copies and Alternative Mappings

18325 TSH For barriers, condition variables, mutexes, and read-write locks, if the process-shared attribute
 18326 is set to `PTHREAD_PROCESS_PRIVATE`, only the synchronization object at the address used to
 18327 initialize it can be used for performing synchronization. The effect of referring to another
 18328 TSH mapping of the same object when locking, unlocking, or destroying the object is undefined. If
 18329 the process-shared attribute is set to `PTHREAD_PROCESS_SHARED`, only the synchronization
 18330 object itself can be used for performing synchronization; however, it need not be referenced at
 18331 the address used to initialize it (that is, another mapping of the same object can be used). The
 18332 effect of referring to a copy of the object when locking, unlocking, or destroying it is undefined.

18333 For spin locks, the above requirements shall apply as if spin locks have a process-shared
 18334 attribute that is set from the `pshared` argument to `pthread_spin_init()`. For semaphores, the above
 18335 requirements shall apply as if semaphores have a process-shared attribute that is set to
 18336 `PTHREAD_PROCESS_PRIVATE` if the `pshared` argument to `sem_init()` is zero and set to
 18337 `PTHREAD_PROCESS_SHARED` if `pshared` is non-zero.

18338 2.10 Sockets

18339 A socket is an endpoint for communication using the facilities described in this section. A socket
 18340 is created with a specific socket type, described in [Section 2.10.6](#) (on page 524), and is associated
 18341 with a specific protocol, detailed in [Section 2.10.3](#) (on page 524). A socket is accessed via a file
 18342 descriptor obtained when the socket is created.

18343 2.10.1 Address Families

18344 All network protocols are associated with a specific address family. An address family provides
 18345 basic services to the protocol implementation to allow it to function within a specific network
 18346 environment. These services may include packet fragmentation and reassembly, routing,
 18347 addressing, and basic transport. An address family is normally comprised of a number of
 18348 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not
 18349 required that an address family support all socket types. An address family may contain
 18350 multiple protocols supporting the same socket abstraction.

18351 [Section 2.10.17](#) (on page 531), [Section 2.10.19](#) (on page 532), and [Section 2.10.20](#) (on page 532),
 18352 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based
 18353 on IPv4, and for Internet protocols based on IPv6.

18354 2.10.2 Addressing

18355 An address family defines the format of a socket address. All network addresses are described
18356 using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of
18357 POSIX.1-2017, <sys/socket.h>. However, each address family imposes finer and more specific
18358 structure, generally defining a structure with fields specific to the address family. The field
18359 *sa_family* in the **sockaddr** structure contains the address family identifier, specifying the format
18360 of the *sa_data* area. The size of the *sa_data* area is unspecified.

18361 2.10.3 Protocols

18362 A protocol supports one of the socket abstractions detailed in [Section 2.10.6](#). Selecting a protocol
18363 involves specifying the address family, socket type, and protocol number to the *socket()* function.
18364 Certain semantics of the basic socket abstractions are protocol-specific. All protocols are
18365 expected to support the basic model for their particular socket type, but may, in addition,
18366 provide non-standard facilities or extensions to a mechanism.

18367 2.10.4 Routing

18368 Sockets provides packet routing facilities. A routing information database is maintained, which
18369 is used in selecting the appropriate network interface when transmitting packets.

18370 2.10.5 Interfaces

18371 Each network interface in a system corresponds to a path through which messages can be sent
18372 and received. A network interface usually has a hardware device associated with it, though
18373 certain interfaces such as the loopback interface, do not.

18374 2.10.6 Socket Types

18375 A socket is created with a specific type, which defines the communication semantics and which
18376 allows the selection of an appropriate communication protocol. Four types are defined:
18377 RS SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET, and SOCK_STREAM. Implementations
18378 may specify additional socket types.

18379 The SOCK_STREAM socket type provides reliable, sequenced, full-duplex octet streams
18380 between the socket and a peer to which the socket is connected. A socket of type
18381 SOCK_STREAM must be in a connected state before any data may be sent or received. Record
18382 boundaries are not maintained; data sent on a stream socket using output operations of one size
18383 may be received using input operations of smaller or larger sizes without loss of data. Data may
18384 be buffered; successful return from an output function does not imply that the data has been
18385 delivered to the peer or even transmitted from the local system. If data cannot be successfully
18386 transmitted within a given time then the connection is considered broken, and subsequent
18387 operations shall fail. A SIGPIPE signal is raised if a thread attempts to send data on a broken
18388 stream (one that is no longer connected), except that the signal is suppressed if the
18389 MSG_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Support for an out-of-
18390 band data transmission facility is protocol-specific.

18391 The SOCK_SEQPACKET socket type is similar to the SOCK_STREAM type, and is also
18392 connection-oriented. The only difference between these types is that record boundaries are
18393 maintained using the SOCK_SEQPACKET type. A record can be sent using one or more output

18394 operations and received using one or more input operations, but a single operation never
18395 transfers parts of more than one record. Record boundaries are visible to the receiver via the
18396 MSG_EOR flag in the received message flags returned by the *recvmsg()* function. It is protocol-
18397 specific whether a maximum record size is imposed.

18398 The SOCK_DGRAM socket type supports connectionless data transfer which is not necessarily
18399 acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or
18400 broadcast) in each output operation, and incoming datagrams may be received from multiple
18401 sources. The source address of each datagram is available when receiving the datagram. An
18402 application may also pre-specify a peer address, in which case calls to output functions that do
18403 not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only
18404 datagrams from that peer shall be received. A datagram must be sent in a single output
18405 operation, and must be received in a single input operation. The maximum size of a datagram is
18406 protocol-specific; with some protocols, the limit is implementation-defined. Output datagrams
18407 may be buffered within the system; thus, a successful return from an output function does not
18408 guarantee that a datagram is actually sent or received. However, implementations should
18409 attempt to detect any errors possible before the return of an output function, reporting any error
18410 by an unsuccessful return value.

18411 RS The SOCK_RAW socket type is similar to the SOCK_DGRAM type. It differs in that it is
18412 normally used with communication providers that underlie those used for the other socket
18413 types. For this reason, the creation of a socket with type SOCK_RAW shall require appropriate
18414 privileges. The format of datagrams sent and received with this socket type generally include
18415 specific protocol headers, and the formats are protocol-specific and implementation-defined.

18416 2.10.7 Socket I/O Mode

18417 The I/O mode of a socket is described by the O_NONBLOCK file status flag which pertains to
18418 the open file description for the socket. This flag is initially off when a socket is created, but may
18419 be set and cleared by the use of the F_SETFL command of the *fcntl()* function.

18420 When the O_NONBLOCK flag is set, certain functions that would normally block until they are
18421 complete shall return immediately.

18422 The *bind()* function initiates an address assignment and shall return without blocking when
18423 O_NONBLOCK is set; if the socket address cannot be assigned immediately, *bind()* shall return
18424 the [EINPROGRESS] error to indicate that the assignment was initiated successfully, but that it
18425 has not yet completed.

18426 The *connect()* function initiates a connection and shall return without blocking when
18427 O_NONBLOCK is set; it shall return the error [EINPROGRESS] to indicate that the connection
18428 was initiated successfully, but that it has not yet completed.

18429 Data transfer operations (the *read()*, *write()*, *send()*, and *recv()* functions) shall complete
18430 immediately, transfer only as much as is available, and then return without blocking, or return
18431 an error indicating that no transfer could be made without blocking.

18432 2.10.8 Socket Owner

18433 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
18434 process group ID using the `F_SETOWN` command of the `fcntl()` function.

18435 2.10.9 Socket Queue Limits

18436 The transmit and receive queue sizes for a socket are set when the socket is created. The default
18437 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
18438 using the `setsockopt()` function.

18439 2.10.10 Pending Error

18440 Errors may occur asynchronously, and be reported to the socket in response to input from the
18441 network protocol. The socket stores the pending error to be reported to a user of the socket at the
18442 next opportunity. The error is returned in response to a subsequent `send()`, `recv()`, or `getsockopt()`
18443 operation on the socket, and the pending error is then cleared.

18444 2.10.11 Socket Receive Queue

18445 A socket has a receive queue that buffers data when it is received by the system until it is
18446 removed by a receive call. Depending on the type of the socket and the communication provider,
18447 the receive queue may also contain ancillary data such as the addressing and other protocol data
18448 associated with the normal data in the queue, and may contain out-of-band or expedited data.
18449 The limit on the queue size includes any normal, out-of-band data, datagram source addresses,
18450 and ancillary data in the queue. The description in this section applies to all sockets, even
18451 though some elements cannot be present in some instances.

18452 The contents of a receive buffer are logically structured as a series of data segments with
18453 associated ancillary data and other information. A data segment may contain normal data or
18454 out-of-band data, but never both. A data segment may complete a record if the protocol
18455 supports records (always true for types `SOCK_SEQPACKET` and `SOCK_DGRAM`). A record
18456 may be stored as more than one segment; the complete record might never be present in the
18457 receive buffer at one time, as a portion might already have been returned to the application, and
18458 another portion might not yet have been received from the communications provider. A data
18459 segment may contain ancillary protocol data, which is logically associated with the segment.
18460 Ancillary data is received as if it were queued along with the first normal data octet in the
18461 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band
18462 data. For the purposes of this section, a datagram is considered to be a data segment that
18463 terminates a record, and that includes a source address as a special type of ancillary data. Data
18464 segments are placed into the queue as data is delivered to the socket by the protocol. Normal
18465 data segments are placed at the end of the queue as they are delivered. If a new segment
18466 contains the same type of data as the preceding segment and includes no ancillary data, and if
18467 the preceding segment does not terminate a record, the segments are logically merged into a
18468 single segment.

18469 The receive queue is logically terminated if an end-of-file indication has been received or a
18470 connection has been terminated. A segment shall be considered to be terminated if another
18471 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other
18472 connection termination has been reported. The last segment in the receive queue shall also be
18473 considered to be terminated while the socket has a pending error to be reported.

18474 A receive operation shall never return data or ancillary data from more than one segment.

18475 2.10.12 Socket Out-of-Band Data State

18476 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed
18477 in the socket receive queue, either at the end of the queue or before all normal data in the queue.
18478 In this case, out-of-band data is returned to an application program by a normal receive call.
18479 Out-of-band data may also be queued separately rather than being placed in the socket receive
18480 queue, in which case it shall be returned only in response to a receive call that requests out-of-
18481 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive
18482 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An
18483 out-of-band data mark is logically an empty data segment that cannot be merged with other
18484 segments in the queue. An out-of-band data mark is never returned in response to an input
18485 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the
18486 first element in the queue. If an out-of-band data mark is the first element in the queue when an
18487 input function is called without the MSG_PEEK option, the mark is removed from the queue
18488 and the following data (if any) is processed as if the mark had not been present.

18489 2.10.13 Connection Indication Queue

18490 Sockets that are used to accept incoming connections maintain a queue of outstanding
18491 connection indications. This queue is a list of connections that are awaiting acceptance by the
18492 application; see *listen()*.

18493 2.10.14 Signals

18494 One category of event at the socket interface is the generation of signals. These signals report
18495 protocol events or process errors relating to the state of the socket. The generation or delivery of
18496 a signal does not change the state of the socket, although the generation of the signal may have
18497 been caused by a state change.

18498 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no
18499 longer able to send (one that is no longer connected), except that the signal is suppressed if the
18500 MSG_NOSIGNAL flag is used in calls to *send()*, *sendto()*, and *sendmsg()*. Regardless of whether
18501 the generation of the signal is suppressed, the send operation shall fail with the [EPIPE] error.

18502 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified
18503 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the
18504 status of the socket is specified in [Section 2.10.17](#) (on page 531), [Section 2.10.19](#) (on page 532),
18505 and [Section 2.10.20](#) (on page 532). Depending on the protocol, the expedited data may or may
18506 not have arrived at the time of signal generation.

18507 2.10.15 Asynchronous Errors

18508 If any of the following conditions occur asynchronously for a socket, the corresponding value
18509 listed below shall become the pending error for the socket:

18510 [ECONNABORTED]

18511 The connection was aborted locally.

18512	[ECONNREFUSED]
18513	For a connection-mode socket attempting a non-blocking connection, the attempt to connect
18514	was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram
18515	was forcefully rejected.
18516	[ECONNRESET]
18517	The peer has aborted the connection.
18518	[EHOSTDOWN]
18519	The destination host has been determined to be down or disconnected.
18520	[EHOSTUNREACH]
18521	The destination host is not reachable.
18522	[EMSGSIZE]
18523	For a connectionless-mode socket, the size of a previously sent datagram prevented
18524	delivery.
18525	[ENETDOWN]
18526	The local network connection is not operational.
18527	[ENETRESET]
18528	The connection was aborted by the network.
18529	[ENETUNREACH]
18530	The destination network is not reachable.

18531 2.10.16 Use of Options

18532 There are a number of socket options which either specialize the behavior of a socket or provide
 18533 useful information. These options may be set at different protocol levels and are always present
 18534 at the uppermost “socket” level.

18535 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions
 18536 allow an application program to customize the behavior and characteristics of a socket to
 18537 provide the desired effect.

18538 All of the options have default values. The type and meaning of these values is defined by the
 18539 protocol level to which they apply. Instead of using the default values, an application program
 18540 may choose to customize one or more of the options. However, in the bulk of cases, the default
 18541 values are sufficient for the application.

18542 Some of the options are used to enable or disable certain behavior within the protocol modules
 18543 (for example, turn on debugging) while others may be used to set protocol-specific information
 18544 (for example, IP time-to-live on all the application’s outgoing packets). As each of the options is
 18545 introduced, its effect on the underlying protocol modules is described.

18546 [Table 2-1](#) shows the value for the socket level.

18547 **Table 2-1** Value of Level for Socket Options

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

18550 [Table 2-2](#) (on page 529) lists those options present at the socket level; that is, when the *level*
 18551 parameter of the *getsockopt()* or *setsockopt()* function is SOL_SOCKET, the types of the option
 18552 value parameters associated with each option, and a brief synopsis of the meaning of the option

18553 value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with
 18554 *setsockopt()* on all types of socket. Options at other protocol levels vary in format and name.

18555 **Table 2-2** Socket-Level Options

Option	Parameter Type	Parameter Meaning
SO_ACCEPTCONN	int	Non-zero indicates that socket listening is enabled (<i>getsockopt()</i> only).
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket (<i>getsockopt()</i> only).
SO_KEEPAIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsend data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type (<i>getsockopt()</i> only).

18586 The SO_ACCEPTCONN option is used only on *getsockopt()*. When this option is specified,
 18587 *getsockopt()* shall report whether socket listening is enabled for the socket. A value of zero shall
 18588 indicate that socket listening is disabled; non-zero that it is enabled. SO_ACCEPTCONN has no
 18589 default value.

18590 The SO_BROADCAST option requests permission to send broadcast datagrams on the socket.
 18591 Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the
 18592 ability to send broadcast datagrams on a socket is disabled.

18593 The SO_DEBUG option enables debugging in the underlying protocol modules. This can be
 18594 useful for tracing the behavior of the underlying protocol modules during normal system
 18595 operation. The semantics of the debug reports are implementation-defined. The default value for
 18596 SO_DEBUG is for debugging to be turned off.

18597 The SO_DONTROUTE option requests that outgoing messages bypass the standard routing
 18598 facilities. The destination must be on a directly-connected network, and messages are directed to
 18599 the appropriate network interface according to the destination address. It is protocol-specific
 18600 whether this option has any effect and how the outgoing network interface is chosen. Support

18601 for this option with each protocol is implementation-defined.

18602 The `SO_ERROR` option is used only on `getsockopt()`. When this option is specified, `getsockopt()`
18603 shall return any pending error on the socket and clear the error status. It shall return a value of 0
18604 if there is no pending error. `SO_ERROR` may be used to check for asynchronous errors on
18605 connected connectionless-mode sockets or for other types of asynchronous errors. `SO_ERROR`
18606 has no default value.

18607 The `SO_KEEPALIVE` option enables the periodic transmission of messages on a connected
18608 socket. The behavior of this option is protocol-specific. On a connection-mode socket for which a
18609 connection has been established, if `SO_KEEPALIVE` is enabled and the connected socket fails to
18610 respond to the keep-alive messages, the connection shall be broken. The default value for
18611 `SO_KEEPALIVE` is zero, specifying that this capability is turned off.

18612 The `SO_LINGER` option controls the action of the interface when unsent messages are queued
18613 on a socket and a `close()` is performed. The details of this option are protocol-specific. If
18614 `SO_LINGER` is enabled, the system shall block the calling thread during `close()` until it can
18615 transmit the data or until the end of the interval indicated by the `l_linger` member, whichever
18616 comes first. If `SO_LINGER` is not specified, and `close()` is issued, the system handles the call in a
18617 way that allows the calling thread to continue as quickly as possible. The default value for
18618 `SO_LINGER` is zero, or off, for the `l_onoff` element of the option value and zero seconds for the
18619 linger time specified by the `l_linger` element.

18620 The `SO_OOBINLINE` option is valid only on protocols that support out-of-band data. The
18621 `SO_OOBINLINE` option requests that out-of-band data be placed in the normal data input
18622 queue as received; it is then accessible using the `read()` or `recv()` functions without the
18623 `MSG_OOB` flag set. The default for `SO_OOBINLINE` is off; that is, for out-of-band data not to be
18624 placed in the normal data input queue.

18625 The `SO_RCVBUF` option requests that the buffer space allocated for receive operations on this
18626 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer
18627 size for high volume connections, or may decrease buffer size to limit the possible backlog of
18628 incoming data. The default value for the `SO_RCVBUF` option value is implementation-defined,
18629 and may vary by protocol.

18630 The `SO_RCVLOWAT` option sets the minimum number of bytes to process for socket input
18631 operations. In general, receive calls block until any (non-zero) amount of data is received, then
18632 return the smaller of the amount available or the amount requested. The default value for
18633 `SO_RCVLOWAT` is 1, and does not affect the general case. If `SO_RCVLOWAT` is set to a larger
18634 value, blocking receive calls normally wait until they have received the smaller of the low water
18635 mark value or the requested amount. Receive calls may still return less than the low water mark
18636 if an error occurs, a signal is caught, or the type of data next in the receive queue is different
18637 from that returned (for example, out-of-band data). As mentioned previously, the default value
18638 for `SO_RCVLOWAT` is 1 byte. It is implementation-defined whether the `SO_RCVLOWAT` option
18639 can be set.

18640 The `SO_RCVTIMEO` option is an option to set a timeout value for input operations. It accepts a
18641 **timeval** structure with the number of seconds and microseconds specifying the limit on how
18642 long to wait for an input operation to complete. If a receive operation has blocked for this much
18643 time without receiving additional data, it shall return with a partial count or `errno` shall be set to
18644 `[EAGAIN]` or `[EWOULDBLOCK]` if no data were received. The default for this option is the
18645 value zero, which indicates that a receive operation will not time out. It is implementation-
18646 defined whether the `SO_RCVTIMEO` option can be set.

18647 The `SO_REUSEADDR` option indicates that the rules used in validating addresses supplied in a
18648 `bind()` should allow reuse of local addresses. Operation of this option is protocol-specific. The
18649 default value for `SO_REUSEADDR` is off; that is, reuse of local addresses is not permitted.

18650 The SO_SNDBUF option requests that the buffer space allocated for send operations on this
 18651 socket be set to the value, in bytes, of the option value. The default value for the SO_SNDBUF
 18652 option value is implementation-defined, and may vary by protocol.

18653 The SO_SNDLOWAT option sets the minimum number of bytes to process for socket output
 18654 operations. Most output operations process all of the data supplied by the call, delivering data to
 18655 the protocol for transmission and blocking as necessary for flow control. Non-blocking output
 18656 operations process as much data as permitted subject to flow control without blocking, but
 18657 process no data if flow control does not allow the smaller of the send low water mark value or
 18658 the entire request to be processed. A *select()* operation testing the ability to write to a socket shall
 18659 return true only if the send low water mark could be processed. The default value for
 18660 SO_SNDLOWAT is implementation-defined and protocol-specific. It is implementation-defined
 18661 whether the SO_SNDLOWAT option can be set.

18662 The SO_SNDTIMEO option is an option to set a timeout value for the amount of time that an
 18663 output function shall block because flow control prevents data from being sent. As noted in
 18664 [Table 2-2](#) (on page 529), the option value is a **timeval** structure with the number of seconds and
 18665 microseconds specifying the limit on how long to wait for an output operation to complete. If a
 18666 send operation has blocked for this much time, it shall return with a partial count or *errno* set to
 18667 [EAGAIN] or [EWOULDBLOCK] if no data were sent. The default for this option is the value
 18668 zero, which indicates that a send operation will not time out. It is implementation-defined
 18669 whether the SO_SNDTIMEO option can be set.

18670 The SO_TYPE option is used only on *getsockopt()*. When this option is specified, *getsockopt()*
 18671 shall return the type of the socket (for example, SOCK_STREAM). This option is useful to
 18672 servers that inherit sockets on start-up. SO_TYPE has no default value.

18673 2.10.17 Use of Sockets for Local UNIX Connections

18674 Support for UNIX domain sockets is mandatory.

18675 UNIX domain sockets provide process-to-process communication in a single system.

18676 2.10.17.1 Headers

18677 The symbolic constant AF_UNIX defined in the `<sys/socket.h>` header is used to identify the
 18678 UNIX domain address family. The `<sys/un.h>` header contains other definitions used in
 18679 connection with UNIX domain sockets. See XBD [Chapter 13](#) (on page 219).

18680 The **sockaddr_storage** structure defined in `<sys/socket.h>` shall be large enough to
 18681 accommodate a **sockaddr_un** structure (see the `<sys/un.h>` header defined in XBD [Chapter 13](#),
 18682 on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be cast as
 18683 pointers to **sockaddr_un** structures and used to access the fields of those structures without
 18684 alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_un** structure, the
 18685 *ss_family* field maps onto the *sun_family* field.

18686 2.10.18 Use of Sockets over Internet Protocols

18687 When a socket is created in the Internet family with a protocol value of zero, the implementation
18688 shall use the protocol listed below for the type of socket created.

18689 SOCK_STREAM IPPROTO_TCP.

18690 SOCK_DGRAM IPPROTO_UDP.

18691 RS SOCK_RAW IPPROTO_RAW.

18692 SOCK_SEQPACKET Unspecified.

18693 RS A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default
18694 protocol for type SOCK_RAW shall be identified in the IP header with the value
18695 IPPROTO_RAW. Applications should not use the default protocol when creating a socket with
18696 type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is
18697 accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.

18698 2.10.19 Use of Sockets over Internet Protocols Based on IPv4

18699 Support for sockets over Internet protocols based on IPv4 is mandatory.

18700 2.10.19.1 Headers

18701 The symbolic constant AF_INET defined in the `<sys/socket.h>` header is used to identify the
18702 IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in
18703 connection with IPv4 Internet sockets. See XBD [Chapter 13](#) (on page 219).

18704 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to
18705 accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in XBD [Chapter](#)
18706 [13](#), on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be
18707 cast as pointers to `sockaddr_in` structures and used to access the fields of those structures
18708 without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in`
18709 structure, the `ss_family` field maps onto the `sin_family` field.

18710 2.10.20 Use of Sockets over Internet Protocols Based on IPv6

18711 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. The
18712 functionality described in this section shall be provided on implementations that support the
18713 IPV6 option (and the rest of this section is not further shaded for this option).

18714 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain
18715 circumstances, also be used in connection with IPv4; see [Section 2.10.20.2](#) (on page 533).

18716 2.10.20.1 Addressing

18717 IPv6 overcomes the addressing limitations of earlier versions by using 128-bit addresses instead
18718 of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

18719 There are three kinds of IPv6 address:

18720 Unicast

18721 Identifies a single interface.

18722 A unicast address can be global, link-local (designed for use on a single link), or site-local

18723 (designed for systems not connected to the Internet). Link-local and site-local addresses
18724 need not be globally unique.

18725 Anycast

18726 Identifies a set of interfaces such that a packet sent to the address can be delivered to any
18727 member of the set.

18728 An anycast address is similar to a unicast address; the nodes to which an anycast address is
18729 assigned must be explicitly configured to know that it is an anycast address.

18730 Multicast

18731 Identifies a set of interfaces such that a packet sent to the address should be delivered to
18732 every member of the set.

18733 An application can send multicast datagrams by simply specifying an IPv6 multicast
18734 address in the *address* argument of *sendto()*. To receive multicast datagrams, an application
18735 must join the multicast group (using *setsockopt()* with `IPV6_JOIN_GROUP`) and must bind
18736 to the socket the UDP port on which datagrams will be received. Some applications should
18737 also bind the multicast group address to the socket, to prevent other datagrams destined to
18738 that port from being delivered to the socket.

18739 A multicast address can be global, node-local, link-local, site-local, or organization-local.

18740 The following special IPv6 addresses are defined:

18741 Unspecified

18742 An address that is not assigned to any interface and is used to indicate the absence of an
18743 address.

18744 Loopback

18745 A unicast address that is not assigned to any interface and can be used by a node to send
18746 packets to itself.

18747 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:

18748 IPv4-compatible addresses

18749 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”
18750 through IPv4.

18751 IPv4-mapped addresses

18752 These are used to represent IPv4 addresses in IPv6 address format; see [Section 2.10.20.2](#).

18753 Note that the unspecified address and the loopback address must not be treated as
18754 IPv4-compatible addresses.

18755 2.10.20.2 Compatibility with IPv4

18756 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,
18757 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the
18758 *getaddrinfo()* function when the specified host has only IPv4 addresses.

18759 Applications can use `AF_INET6` sockets to open TCP connections to IPv4 nodes, or send UDP
18760 packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped
18761 IPv6 address, and passing that address, within a `sockaddr_in6` structure, in the *connect()*,
18762 *sendto()*, or *sendmsg()* function. When applications use `AF_INET6` sockets to accept TCP
18763 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return
18764 the peer’s address to the application in the *accept()*, *recvfrom()*, *recvmsg()*, or *getpeername()*
18765 function using a `sockaddr_in6` structure encoded this way. If a node has an IPv4 address, then
18766 the implementation shall allow applications to communicate using that address via an

18767 AF_INET6 socket. In such a case, the address will be represented at the API by the
 18768 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an AF_INET6
 18769 socket bound to **in6addr_any** to receive inbound connections and packets destined to one of the
 18770 node's IPv4 addresses.

18771 An application can use AF_INET6 sockets to bind to a node's IPv4 address by specifying the
 18772 address as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure in the *bind()* function. For
 18773 an AF_INET6 socket bound to a node's IPv4 address, the system shall return the address in the
 18774 *getsockname()* function as an IPv4-mapped IPv6 address in a **sockaddr_in6** structure.

18775 2.10.20.3 Interface Identification

18776 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;
 18777 zero is not used. There may be gaps so that there is no current interface for a particular positive
 18778 index. Each interface also has a unique implementation-defined name.

18779 2.10.20.4 Options

18780 The following options apply at the IPPROTO_IPV6 level:

18781 IPV6_JOIN_GROUP

18782 When set via *setsockopt()*, it joins the application to a multicast group on an interface
 18783 (identified by its index) and addressed by a given multicast address, enabling packets sent
 18784 to that address to be read via the socket. If the interface index is specified as zero, the
 18785 system selects the interface (for example, by looking up the address in a routing table and
 18786 using the resulting interface).

18787 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

18788 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

18789 IPV6_LEAVE_GROUP

18790 When set via *setsockopt()*, it removes the application from the multicast group on an
 18791 interface (identified by its index) and addressed by a given multicast address.

18792 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

18793 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

18794 IPV6_MULTICAST_HOPS

18795 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
 18796 socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the
 18797 IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set
 18798 via *setsockopt()* and read via *getsockopt()*.

18799 The parameter type of this option is a pointer to an **int**. (Default value: 1)

18800 IPV6_MULTICAST_IF

18801 The index of the interface to be used for outgoing multicast packets. It can be set via
 18802 *setsockopt()* and read via *getsockopt()*. If the interface index is specified as zero, the system
 18803 selects the interface (for example, by looking up the address in a routing table and using the
 18804 resulting interface).

18805 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

18806 IPV6_MULTICAST_LOOP

18807 This option controls whether outgoing multicast packets should be delivered back to the
 18808 local application when the sending interface is itself a member of the destination multicast

18809 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
18810 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

18811 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean
18812 value. (Default value: 1)

18813 IPV6_UNICAST_HOPS

18814 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
18815 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to
18816 set a value less than -1 or greater than 255 shall result in an [EINVAL] error. This option can
18817 be set via *setsockopt()* and read via *getsockopt()*.

18818 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

18819 IPV6_V6ONLY

18820 This socket option restricts AF_INET6 sockets to IPv6 communications only. AF_INET6
18821 sockets may be used for both IPv4 and IPv6 communications. Some applications may want
18822 to restrict their use of an AF_INET6 socket to IPv6 communications only. For these
18823 applications, the IPV6_V6ONLY socket option is defined. When this option is turned on, the
18824 socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6-level
18825 option.

18826 The parameter type of this option is a pointer to an **int** which is used as a Boolean value.
18827 (Default value: 0)

18828 An [EOPNOTSUPP] error shall result if IPV6_JOIN_GROUP or IPV6_LEAVE_GROUP is used
18829 with *getsockopt()*.

18830 2.10.20.5 Headers

18831 The symbolic constant AF_INET6 is defined in the `<sys/socket.h>` header to identify the IPv6
18832 Internet address family. See XBD [Chapter 13](#) (on page 219).

18833 The **sockaddr_storage** structure defined in `<sys/socket.h>` shall be large enough to
18834 accommodate a **sockaddr_in6** structure (see the `<netinet/in.h>` header defined in XBD [Chapter](#)
18835 [13](#), on page 219) and shall be aligned at an appropriate boundary so that pointers to it can be
18836 cast as pointers to **sockaddr_in6** structures and used to access the fields of those structures
18837 without alignment problems. When a **sockaddr_storage** structure is cast as a **sockaddr_in6**
18838 structure, the *ss_family* field maps onto the *sin6_family* field.

18839 The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in
18840 connection with IPv6 Internet sockets; see XBD [Chapter 13](#) (on page 219).

18841 2.11 Tracing

18842 OB TRC This section describes extensions to support tracing of user applications. The functionality
18843 described in this section is dependent on support of the Trace option (and the rest of this section
18844 is not further shaded for this option).

18845 The tracing facilities defined in POSIX.1-2017 allow a process to select a set of trace event types,
18846 to activate a trace stream of the selected trace events as they occur in the flow of execution, and
18847 to retrieve the recorded trace events.

18848 The tracing operation relies on three logically different components: the traced process, the
18849 controller process, and the analyzer process. During the execution of the traced process, when a
18850 trace point is reached, a trace event is recorded into the trace streams created for that process in
18851 which the associated trace event type identifier is not being filtered out. The controller process
18852 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 18853 Initialize the attributes of a trace stream
- 18854 Create the trace stream (for a specified traced process) using those attributes
- 18855 Start and stop tracing for the trace stream
- 18856 Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 18857 Shut a trace stream down

18858 These operations can be done for an active trace stream. The analyzer process retrieves the
18859 traced events either at runtime, when the trace stream has not yet been shut down, but is still
18860 recording trace events; or after opening a trace log that had been previously recorded and shut
18861 down. These three logically different operations can be performed by the same process, or can
18862 be distributed into different processes.

18863 A trace stream identifier can be created by a call to *posix_trace_create()*,
18864 *posix_trace_create_withlog()*, or *posix_trace_open()*. The *posix_trace_create()* and
18865 *posix_trace_create_withlog()* functions should be used by a controller process. The
18866 *posix_trace_open()* should be used by an analyzer process.

18867 The tracing functions can serve different purposes. One purpose is debugging the possibly pre-
18868 instrumented code, while another is post-mortem fault analysis. These two potential uses differ
18869 in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and
18870 permits focusing on expected information; while the second needs comprehensive trace
18871 capabilities in order to be able to record all types of information.

18872 The events to be traced belong to two classes:

- 18873 1. User trace events (generated by the application instrumentation)
- 18874 2. System trace events (generated by the operating system)

18875 The trace interface defines several system trace event types associated with control of and
18876 operation of the trace stream. This small set of system trace events includes the minimum
18877 required to interpret correctly the trace event information present in the stream. Other desirable
18878 system trace events for some particular application profile may be implemented and are
18879 encouraged; for example, process and thread scheduling, signal occurrence, and so on.

18880 Each traced process shall have a mapping of the trace event names to trace event type identifiers
18881 that have been defined for that process. Each active trace stream shall have a mapping that
18882 incorporates all the trace event type identifiers predefined by the trace system plus all the
18883 mappings of trace event names to trace event type identifiers of the processes that are being
18884 traced into that trace stream. These mappings are defined from the instrumented application by
18885 calling the *posix_trace_eventid_open()* function and from the controller process by calling the

18886 *posix_trace_trid_eventid_open()* function. For a pre-recorded trace stream, the list of trace event
18887 types is obtained from the pre-recorded trace log.

18888 The last data modification and file status change timestamps of a file associated with an active
18889 trace stream shall be marked for update every time any of the tracing operations modifies that
18890 file.

18891 The last data access timestamp of a file associated with a trace stream shall be marked for
18892 update every time any of the tracing operations causes data to be read from that file.

18893 Results are undefined if the application performs any operation on a file descriptor associated
18894 with an active or pre-recorded trace stream until *posix_trace_shutdown()* or *posix_trace_close()*
18895 is called for that trace stream. Results are also undefined if the analyzer process and the traced
18896 process do not share the same programming environment (see *c99*, Programming Environments
18897 in the Shell and Utilities volume of POSIX.1-2017).

18898 The main purpose of this option is to define a complete set of functions and concepts that allow
18899 a conforming application to be traced from creation to termination, whatever its realtime
18900 constraints and properties.

18901 **2.11.1 Tracing Data Definitions**18902 *2.11.1.1 Structures*

18903 The **<trace.h>** header shall define the *posix_trace_status_info* and *posix_trace_event_info* structures
 18904 described below. Implementations may add extensions to these structures.

18905 **posix_trace_status_info Structure**

18906 To facilitate control of a trace stream, information about the current state of an active trace
 18907 stream can be obtained dynamically. This structure is returned by a call to the
 18908 *posix_trace_get_status()* function.

18909 The **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following
 18910 members:

Member Type	Member Name	Description
int	<i>posix_stream_status</i>	The operating mode of the trace stream.
int	<i>posix_stream_full_status</i>	The full status of the trace stream.
int	<i>posix_stream_overrun_status</i>	Indicates whether trace events were lost in the trace stream.

18916 If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info**
 18917 structure defined in **<trace.h>** shall contain at least the following additional members:

Member Type	Member Name	Description
int	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
int	<i>posix_stream_flush_error</i>	Indicates whether any error occurred during the last flush operation.
int	<i>posix_log_overrun_status</i>	Indicates whether trace events were lost in the trace log.
int	<i>posix_log_full_status</i>	The full status of the trace log.

18925 The *posix_stream_status* member indicates the operating mode of the trace stream and shall have
 18926 one of the following values defined by manifest constants in the **<trace.h>** header:

18927 POSIX_TRACE_RUNNING

18928 Tracing is in progress; that is, the trace stream is accepting trace events.

18929 POSIX_TRACE_SUSPENDED

18930 The trace stream is not accepting trace events. The tracing operation has not yet started or
 18931 has stopped, either following a *posix_trace_stop()* function call or because the trace resources
 18932 are exhausted.

18933 The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have
 18934 one of the following values defined by manifest constants in the **<trace.h>** header:

18935 POSIX_TRACE_FULL

18936 The space in the trace stream for trace events is exhausted.

18937 POSIX_TRACE_NOT_FULL

18938 There is still space available in the trace stream.

18939 The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates
 18940 the actual status of the stream. The status shall be interpreted as follows:

- 18941 POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL
 18942 This status combination indicates that tracing is in progress, and there is space available for
 18943 recording more trace events.
- 18944 POSIX_TRACE_RUNNING and POSIX_TRACE_FULL
 18945 This status combination indicates that tracing is in progress and that the trace stream is full
 18946 of trace events. This status combination cannot occur unless the *stream-full-policy* is set to
 18947 POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving
 18948 time window of prior trace events, and some older trace events may have been overwritten
 18949 and thus lost.
- 18950 POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL
 18951 This status combination indicates that tracing has not yet been started, has been stopped by
 18952 the *posix_trace_stop()* function, or has been cleared by the *posix_trace_clear()* function.
- 18953 POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL
 18954 This status combination indicates that tracing has been stopped by the implementation
 18955 because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace
 18956 resources were exhausted, or that the trace stream was stopped by the function
 18957 *posix_trace_stop()* at a time when trace resources were exhausted.
- 18958 The *posix_stream_overrun_status* member indicates whether trace events were lost in the trace
 18959 stream, and shall have one of the following values defined by manifest constants in the
 18960 **<trace.h>** header:
- 18961 POSIX_TRACE_OVERRUN
 18962 At least one trace event was lost and thus was not recorded in the trace stream.
- 18963 POSIX_TRACE_NO_OVERRUN
 18964 No trace events were lost.
- 18965 When the corresponding trace stream is created, the *posix_stream_overrun_status* member shall be
 18966 set to POSIX_TRACE_NO_OVERRUN.
- 18967 Whenever an overrun occurs, the *posix_stream_overrun_status* member shall be set to
 18968 POSIX_TRACE_OVERRUN.
- 18969 An overrun occurs when:
- 18970 The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.
- 18971 The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event
 18972 is generated.
- 18973 If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one
 18974 trace event is lost while flushing the trace stream to the trace log.
- 18975 The *posix_stream_overrun_status* member is reset to zero after its value is read.
- 18976 If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*,
 18977 *posix_stream_flush_error*, *posix_log_overrun_status*, and *posix_log_full_status* members are defined
 18978 as follows; otherwise, they are undefined.
- 18979 The *posix_stream_flush_status* member indicates whether a flush operation is being performed
 18980 and shall have one of the following values defined by manifest constants in the header
 18981 **<trace.h>**:
- 18982 POSIX_TRACE_FLUSHING
 18983 The trace stream is currently being flushed to the trace log.

- 18984 POSIX_TRACE_NOT_FLUSHING
18985 No flush operation is in progress.
- 18986 The *posix_stream_flush_status* member shall be set to POSIX_TRACE_FLUSHING if a flush
18987 operation is in progress either due to a call to the *posix_trace_flush()* function (explicit or caused
18988 by a trace stream shutdown operation) or because the trace stream has become full with the
18989 *stream-full-policy* attribute set to POSIX_TRACE_FLUSH. The *posix_stream_flush_status* member
18990 shall be set to POSIX_TRACE_NOT_FLUSHING if no flush operation is in progress.
- 18991 The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If
18992 an error occurred during a previous flushing operation, the *posix_stream_flush_error* member
18993 shall be set to the value of the first error that occurred. If more than one error occurs while
18994 flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is
18995 reset to zero after its value is read.
- 18996 The *posix_log_outrun_status* member indicates whether trace events were lost in the trace log,
18997 and shall have one of the following values defined by manifest constants in the **<trace.h>**
18998 header:
- 18999 POSIX_TRACE_OVERRUN
19000 At least one trace event was lost.
- 19001 POSIX_TRACE_NO_OVERRUN
19002 No trace events were lost.
- 19003 When the corresponding trace stream is created, the *posix_log_outrun_status* member shall be set
19004 to POSIX_TRACE_NO_OVERRUN. Whenever an overrun occurs, this status shall be set to
19005 POSIX_TRACE_OVERRUN. The *posix_log_outrun_status* member is reset to zero after its value
19006 is read.
- 19007 The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of
19008 the following values defined by manifest constants in the **<trace.h>** header:
- 19009 POSIX_TRACE_FULL
19010 The space in the trace log is exhausted.
- 19011 POSIX_TRACE_NOT_FULL
19012 There is still space available in the trace log.
- 19013 The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either
19014 POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.
- 19015 For an active trace stream without log, that is created by the *posix_trace_create()* function, the
19016 *posix_log_outrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the
19017 *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.
- 19018 **posix_trace_event_info Structure**
- 19019 The trace event structure **posix_trace_event_info** contains the information for one recorded
19020 trace event. This structure is returned by the set of functions *posix_trace_getnext_event()*,
19021 *posix_trace_timedgetnext_event()*, and *posix_trace_trygetnext_event()*.
- 19022 The **posix_trace_event_info** structure defined in **<trace.h>** shall contain at least the following
19023 members:

Member Type	Member Name	Description
trace_event_id_t	<i>posix_event_id</i>	Trace event type identification.
pid_t	<i>posix_pid</i>	Process ID of the process that generated the trace event.
void *	<i>posix_prog_address</i>	Address at which the trace point was invoked.
int	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
struct timespec	<i>posix_timestamp</i>	Time at which the trace event was generated.

In addition, the **posix_trace_event_info** structure defined in **<trace.h>** shall contain the following additional member:

Member Type	Member Name	Description
pthread_t	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

The *posix_event_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix_trace_trid_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*, or *posix_trace_eventid_open()*. The name of the trace event type can be obtained by calling *posix_trace_eventid_get_name()*.

The *posix_pid* is the process identifier of the traced process which generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix_pid* member shall be set to zero.

For a user trace event, the *posix_prog_address* member is the process mapped address of the point at which the associated call to the *posix_trace_event()* function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix_prog_address* member shall be the address of the process at the point where the call to that system service was made.

The *posix_truncation_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix_trace_getnext_event()* function). The *posix_truncation_status* member shall have one of the following values defined by manifest constants in the **<trace.h>** header:

POSIX_TRACE_NOT_TRUNCATED
All the traced data is available.

POSIX_TRACE_TRUNCATED_RECORD
Data was truncated at the time the trace event was generated.

POSIX_TRACE_TRUNCATED_READ
Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the POSIX_TRACE_TRUNCATED_RECORD status.

The *posix_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix_trace_attr_getclockres()* function.

The *posix_thread_id* member is the identifier of the thread that generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace

19070 event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

19071 2.11.1.2 Trace Stream Attributes

19072 Trace streams have attributes that compose the **posix_trace_attr_t** trace stream attributes object.
19073 This object shall contain at least the following attributes:

19074 The *generation-version* attribute identifies the origin and version of the trace system.

19075 The *trace-name* attribute is a character string defined by the trace controller, and that
19076 identifies the trace stream.

19077 The *creation-time* attribute represents the time of the creation of the trace stream.

19078 The *clock-resolution* attribute defines the clock resolution of the clock used to generate
19079 timestamps.

19080 The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly
19081 reserved for the trace events.

19082 The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its
19083 value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.

19084 The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

19085 In addition, if the Trace option and the Trace Inherit option are both supported, the
19086 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
19087 attributes:

19088 The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing
19089 in its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or
19090 `POSIX_TRACE_CLOSE_FOR_CHILD`.

19091 In addition, if the Trace option and the Trace Log option are both supported, the
19092 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
19093 attribute:

19094 If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the
19095 `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum
19096 size in bytes of the trace log associated with an active trace stream. Other stream data—for
19097 example, trace attribute values ‡shall not be included in this size.

19098 The *log-full-policy* attribute defines the policy of a trace log associated with an active trace
19099 stream to be `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or
19100 `POSIX_TRACE_APPEND`.

19101 2.11.2 Trace Event Type Definitions

19102 2.11.2.1 System Trace Event Type Definitions

19103 The following system trace event types, defined in the **<trace.h>** header, track the invocation of
19104 the trace operations:

19105 `POSIX_TRACE_START` shall be associated with a trace start operation.

- 19106 POSIX_TRACE_STOP shall be associated with a trace stop operation.
- 19107 If the Trace Event Filter option is supported, POSIX_TRACE_FILTER shall be associated
19108 with a trace event type filter change operation.
- 19109 The following system trace event types, defined in the **<trace.h>** header, report operational trace
19110 events:
- 19111 POSIX_TRACE_OVERFLOW shall mark the beginning of a trace overflow condition.
- 19112 POSIX_TRACE_RESUME shall mark the end of a trace overflow condition.
- 19113 If the Trace Log option is supported, POSIX_TRACE_FLUSH_START shall mark the
19114 beginning of a flush operation.
- 19115 If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of
19116 a flush operation.
- 19117 If an implementation-defined trace error condition is reported, it shall be marked
19118 POSIX_TRACE_ERROR.
- 19119 The interpretation of a trace stream or a trace log by a trace analyzer process relies on the
19120 information recorded for each trace event, and also on system trace events that indicate the
19121 invocation of trace control operations and trace system operational trace events.
- 19122 The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows
19123 during which the trace stream is running.
- 19124 The POSIX_TRACE_STOP trace event with an associated data that is equal to zero
19125 indicates a call of the function *posix_trace_stop()*.
- 19126 The POSIX_TRACE_STOP trace event with an associated data that is different from zero
19127 indicates an automatic stop of the trace stream (see the definition of the
19128 *posix_trace_attr_getstreamfullpolicy()* function in *posix_trace_attr_getinherited()*).
- 19129 The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed
19130 while the trace stream was running.
- 19131 The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-
19132 defined internal error of the trace system occurred.
- 19133 The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the
19134 timestamp of the first trace event overwritten. This is an indication that some generated trace
19135 events have been lost.
- 19136 The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the
19137 timestamp of the first valid trace event reported after the overflow condition ends and shall be
19138 reported before this first valid trace event. This is an indication that the trace system is reliably
19139 recording trace events after an overflow condition.
- 19140 Each of these trace event types shall be defined by a constant trace event name and a
19141 **trace_event_id_t** constant; trace event data is associated with some of these trace events.
- 19142 If the Trace option is supported and the Trace Event Filter option and the Trace Log option are
19143 not supported, the following predefined system trace events in [Table 2-3](#) (on page 544) shall be
19144 defined:

19145

Table 2-3 Trace Option: System Trace Events

19146

19147

19148

19149

19150

19151

19152

19153

19154

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

19155

19156

19157

If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log option is not supported, the following predefined system trace events in [Table 2-4](#) shall be defined:

19158

Table 2-4 Trace and Trace Event Filter Options: System Trace Events

19159

19160

19161

19162

19163

19164

19165

19166

19167

19168

19169

19170

19171

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

19172

19173

19174

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in [Table 2-5](#) (on page 545) shall be defined:

19175

Table 2-5 Trace and Trace Log Options: System Trace Events

19176

19177

19178

19179

19180

19181

19182

19183

19184

19185

19186

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

19187

19188

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in [Table 2-6](#) shall be defined:

19189

Table 2-6 Trace, Trace Log, and Trace Event Filter Options: System Trace Events

19190

19191

19192

19193

19194

19195

19196

19197

19198

19199

19200

19201

19202

19203

19204

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

19205 2.11.2.2 User Trace Event Type Definitions

19206

19207

19208

19209

19210

The user trace event `POSIX_TRACE_UNNAMED_USEREVENT` is defined in the `<trace.h>` header. If the limit of per-process user trace event names represented by `{TRACE_USER_EVENT_MAX}` has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

19211

The following predefined user trace event in [Table 2-7](#) (on page 546) shall be defined:

19212

Table 2-7 Trace Option: User Trace Event

19213

19214

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

19215 **2.11.3 Trace Functions**

19216

19217

19218

19219

19220

19221

The trace interface is built and structured to improve portability through use of trace data of opaque type. The object-oriented approach for the manipulation of trace attributes and trace event type identifiers requires definition of many constructor and selector functions which operate on these opaque types. Also, the trace interface must support several different tracing roles. To facilitate reading the trace interface, the trace functions are grouped into small functional sets supporting the three different roles:

19222

19223

A trace controller process requires functions to set up and customize all the resources needed to run a trace stream, including:

19224

‡ trace attribute initialization and destruction (*posix_trace_attr_init()*)

19225

‡ trace identification information manipulation (*posix_trace_attr_getgenversion()*)

19226

‡ trace system behavior modification (*posix_trace_attr_getinherited()*)

19227

‡ trace stream and trace log size set (*posix_trace_attr_getmaxusereventsize()*)

19228

‡ trace stream creation, flush, and shutdown (*posix_trace_create()*)

19229

‡ trace stream and trace log clear (*posix_trace_clear()*)

19230

‡ trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)

19231

‡ trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)

19232

‡ trace event type set manipulation (*posix_trace_eventset_empty()*)

19233

‡ trace event type filter set (*posix_trace_set_filter()*)

19234

‡ trace stream start and stop (*posix_trace_start()*)

19235

‡ trace stream information and status read (*posix_trace_get_attr()*)

19236

A traced process requires functions to instrument trace points:

19237

‡ trace event type identifiers definition and trace points insertion (*posix_trace_event()*)

19238

19239

A trace analyzer process requires functions to retrieve information from a trace stream and trace log:

19240

‡ trace identification information read (*posix_trace_attr_getgenversion()*)

19241

‡ trace system behavior information read (*posix_trace_attr_getinherited()*)

19242

‡ trace stream and trace log size get (*posix_trace_attr_getmaxusereventsize()*)

19243

‡ trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)

19244

‡ trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)

19245

‡ trace log open, rewind, and close (*posix_trace_open()*)

19246 †read stream information and status read (*posix_trace_get_attr()*)

19247 †read event read (*posix_trace_getnext_event()*)

19248 2.12 Data Types

19249 2.12.1 Defined Types

19250 All of the data types used by various functions are defined by the implementation. The
 19251 following table describes some of these types. Other types referenced in the description of a
 19252 function, not mentioned here, can be found in the appropriate header for that function.

19253	Defined Type	Description
19254	cc_t	Type used for terminal special characters.
19255	clock_t	Integer or real-floating type used for processor times, as defined in the ISO C standard.
19256		
19257	clockid_t	Used for clock ID type in some timer functions.
19258	dev_t	Integer type used for device numbers.
19259	DIR	Type representing a directory stream.
19260	div_t	Structure type returned by the <i>div()</i> function.
19261	FILE	Structure containing information about a file.
19262	glob_t	Structure type used in pathname pattern matching.
19263	fpos_t	Type containing all information needed to specify uniquely every position within a file.
19264		
19265	gid_t	Integer type used for group IDs.
19266	iconv_t	Type used for conversion descriptors.
19267	id_t	Integer type used as a general identifier; can be used to contain at least the largest of a pid_t , uid_t , or gid_t .
19268		
19269	ino_t	Unsigned integer type used for file serial numbers.
19270	key_t	Arithmetic type used for XSI interprocess communication.
19271	ldiv_t	Structure type returned by the <i>ldiv()</i> function.
19272	mode_t	Integer type used for file attributes.
19273	mqd_t	Used for message queue descriptors.
19274	nfds_t	Integer type used for the number of file descriptors.
19275	nlink_t	Integer type used for link counts.
19276	off_t	Signed integer type used for file sizes.
19277	pid_t	Signed integer type used for process and process group IDs.
19278	pthread_attr_t	Used to identify a thread attribute object.
19279	pthread_cond_t	Used for condition variables.
19280	pthread_condattr_t	Used to identify a condition attribute object.
19281	pthread_key_t	Used for thread-specific data keys.
19282	pthread_mutex_t	Used for mutexes.
19283	pthread_mutexattr_t	Used to identify a mutex attribute object.
19284	pthread_once_t	Used for dynamic package initialization.
19285	pthread_rwlock_t	Used for read-write locks.
19286	pthread_rwlockattr_t	Used for read-write lock attributes.
19287	pthread_t	Used to identify a thread.

Defined Type	Description
19288 19289 19290 19291 19292 19293 19294 19295 19296 19297 19298 19299 19300 19301 19302 19303 19304 19305 19306 19307 19308 19309 19310 19311 19312 19313 19314 19315 19316 19317	<p>ptrdiff_t Signed integer type of the result of subtracting two pointers.</p> <p>regex_t Structure type used in regular expression matching.</p> <p>regmatch_t Structure type used in regular expression matching.</p> <p>rlim_t Unsigned integer type used for limit values, to which objects of type int and off_t can be cast without loss of value.</p> <p>sem_t Type used in performing semaphore operations.</p> <p>sig_atomic_t Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.</p> <p>sigset_t Integer or structure type of an object used to represent sets of signals.</p> <p>size_t Unsigned integer type used for size of objects.</p> <p>speed_t Type used for terminal baud rates.</p> <p>ssize_t Signed integer type used for a count of bytes or an error indication.</p> <p>suseconds_t Signed integer type used for time in microseconds.</p> <p>tcfld_t Type used for terminal modes.</p> <p>time_t Integer type used for time in seconds, as defined in the ISO C standard.</p> <p>timer_t Used for timer ID returned by the <i>timer_create()</i> function.</p> <p>uid_t Integer type used for user IDs.</p> <p>va_list Type used for traversing variable argument lists.</p> <p>wchar_t Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.</p> <p>wctype_t Scalar type which represents a character class descriptor.</p> <p>wint_t Integer type capable of storing any valid value of wchar_t or WEOF.</p> <p>wordexp_t Structure type used in word expansion.</p>

19318 2.12.2 The char Type

19319 The type **char** is defined as a single byte; see XBD [Chapter 3](#) (on page 33) (Byte and Character).

19320 2.13 Status Information

19321 Status information is data associated with a process detailing a change in the state of the process.
19322 It shall consist of:

19323 The state the process transitioned into (*stopped*, *continued*, or *terminated*)

19324 The information necessary to populate the **siginfo_t** structure provided by *waitid()*

19325 If the new state is *terminated*:

19326 ‡ The low-order 8 bits of the status argument that the process passed to *_Exit()*, *_exit()*,
19327 or *exit()*, or the low-order 8 bits of the value the process returned from *main()*

19328 Note that these 8 bits are part of the complete value that is used to set the *si_status*
19329 member of the **siginfo_t** structure provided by *waitid()*

19330 ‡ Whether the process terminated due to the receipt of a signal that was not caught
 19331 and, if so, the number of the signal that caused the termination of the process

19332 If the new state is *stopped*:

19333 ‡ Number of the signal that caused the process to stop

19334 A process might not have any status information (such as immediately after a process has
 19335 started).

19336 Status information for a process shall be generated (made available to the parent process) when
 19337 the process stops, continues, or terminates except in the following case:

19338 If the parent process sets the action for the SIGCHLD signal to SIG_IGN, or if the parent
 19339 sets the SA_NOCLDWAIT flag for the SIGCHLD signal action, process termination shall
 19340 not generate new status information but shall cause any existing status information for the
 19341 process to be discarded.

19342 If new status information is generated, and the process already had status information, the
 19343 existing status information shall be discarded and replaced with the new status information.

19344 Only the process' parent process can obtain the process' status information. The parent obtains a
 19345 child's status information by calling *wait()*, *waitid()*, or *waitpid()*. Except when *waitid()* is called
 19346 with the WNOWAIT flag set in the *options* argument, the status information obtained by a wait
 19347 function shall be consumed (discarded) by that wait function; no two calls to *wait()*, *waitid()*
 19348 (without WNOWAIT), or *waitpid()* shall obtain the same status information.

19349 When status information becomes available to the parent process and more than one thread in
 19350 the parent process is waiting for the status information (blocked in a call to *wait()*, *waitid()*, or
 19351 *waitpid()* with arguments that would match the status information):

19352 If none of the matching threads is in a call to *waitid()* with the WNOWAIT flag set in the
 19353 *options* argument, the thread that obtains the status information is unspecified.

19354 Otherwise (at least one of the matching threads is in a call to *waitid()* with the WNOWAIT
 19355 flag set), the matching thread or threads that obtain the status information is unspecified
 19356 except that at least one of the matching threads shall obtain the status information and at
 19357 most one of the matching threads that are not calling *waitid()* with the WNOWAIT flag set
 19358 shall obtain it.


19359 2.14 File Descriptor Allocation

19360 All functions that open one or more file descriptors shall, unless specified otherwise, atomically
 19361 allocate the lowest numbered available (that is, not already open in the calling process) file
 19362 descriptor at the time of each allocation. Where a single function allocates two file descriptors
 19363 (for example, *pipe()* or *socketpair()*), the allocations may be independent and therefore
 19364 applications should not expect them to have adjacent values or depend on which has the higher
 19365 value.

19366

Chapter 3

19367



System Interfaces

19368

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

19369

19370 **NAME**
19371 FD_CLR — macros for synchronous I/O multiplexing

19372 **SYNOPSIS**
19373 #include <sys/select.h>

19374 void FD_CLR(int *fd*, fd_set **fdset*);
19375 int FD_ISSET(int *fd*, fd_set **fdset*);
19376 void FD_SET(int *fd*, fd_set **fdset*);
19377 void FD_ZERO(fd_set **fdset*);

19378 **DESCRIPTION**
19379 Refer to *pselect()*.

19380 **NAME**19381 `_Exit, _exit` — terminate a process19382 **SYNOPSIS**19383 `#include <stdlib.h>`19384 `void _Exit(int status);`19385 `#include <unistd.h>`19386 `void _exit(int status);`19387 **DESCRIPTION**19388 CX For `_Exit()`: The functionality described on this reference page is aligned with the ISO C
19389 standard. Any conflict between the requirements described here and the ISO C standard is
19390 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.19391 CX The value of *status* may be 0, `EXIT_SUCCESS`, `EXIT_FAILURE`, or any other value, though only
19392 the least significant 8 bits (that is, *status* & 0377) shall be available from `wait()` and `waitpid()`; the
19393 full value shall be available from `waitid()` and in the `siginfo_t` passed to a signal handler for
19394 `SIGCHLD`.19395 CX The `_Exit()` and `_exit()` functions shall be functionally equivalent.19396 CX The `_Exit()` and `_exit()` functions shall not call functions registered with `atexit()` nor any
19397 CX registered signal handlers. Open streams shall not be flushed. Whether open streams are
19398 closed (without flushing) is implementation-defined. Finally, the calling process shall be
19399 terminated with the consequences described below.19400 **Consequences of Process Termination**

19401 CX Process termination caused by any reason shall have the following consequences:

19402 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.
19403 However, functionality relating to the XSI option is shaded.19404 All of the file descriptors, directory streams, conversion descriptors, and message catalog
19405 descriptors open in the calling process shall be closed.19406 XSI If the parent process of the calling process has set its `SA_NOCLDWAIT` flag or has set the
19407 action for the `SIGCHLD` signal to `SIG_IGN`:19408 † The process' status information (see [Section 2.13](#), on page 548), if any, shall be
19409 discarded.19410 † The lifetime of the calling process shall end immediately. If `SA_NOCLDWAIT` is set,
19411 it is implementation-defined whether a `SIGCHLD` signal is sent to the parent process.19412 † If a thread in the parent process of the calling process is blocked in `wait()`, `waitpid()`,
19413 or `waitid()`, and the parent process has no remaining child processes in the set of
19414 waited-for children, the `wait()`, `waitid()`, or `waitpid()` function shall fail and set `errno`
19415 to `[ECHILD]`.19416 **Otherwise:**19417 † The status information (see [Section 2.13](#), on page 548) shall be generated.19418 † The calling process shall be transformed into a zombie process. Its status information
19419 shall be made available to the parent process until the process' lifetime ends.

19420 ‡ The process' lifetime shall end once its parent obtains the process' status information
19421 via a currently-blocked or future call to *wait()*, *waitid()* (without *WNOWAIT*), or
19422 *waitpid()*.

19423 ‡ If one or more threads in the parent process of the calling process is blocked in a call
19424 to *wait()*, *waitid()*, or *waitpid()* awaiting termination of the process, one (or, if any are
19425 calling *waitid()* with *WNOWAIT*, possibly more) of these threads shall obtain the
19426 process' status information as specified in [Section 2.13](#) (on page 548) and become
19427 unblocked.

19428 ‡ *SIGCHLD* shall be sent to the parent process.

19429 Termination of a process does not directly terminate its children. The sending of a
19430 *SIGHUP* signal as described below indirectly terminates children in some circumstances.

19431 The parent process ID of all of the existing child processes and zombie processes of the
19432 calling process shall be set to the process ID of an implementation-defined system process.
19433 That is, these processes shall be inherited by a special system process.

19434 XSI Each attached shared-memory segment is detached and the value of *shm_nattch* (see
19435 *shmget()*) in the data structure associated with its shared memory ID shall be decremented
19436 by 1.

19437 XSI For each semaphore for which the calling process has set a *semadj* value (see *semop()*), that
19438 value shall be added to the *semval* of the specified semaphore.

19439 If the process is a controlling process, the *SIGHUP* signal shall be sent to each process in
19440 the foreground process group of the controlling terminal belonging to the calling process.

19441 If the process is a controlling process, the controlling terminal associated with the session
19442 shall be disassociated from the session, allowing it to be acquired by a new controlling
19443 process.

19444 If the exit of the process causes a process group to become orphaned, and if any member of
19445 the newly-orphaned process group is stopped, then a *SIGHUP* signal followed by a
19446 *SIGCONT* signal shall be sent to each process in the newly-orphaned process group.

19447 All open named semaphores in the calling process shall be closed as if by appropriate calls
19448 to *sem_close()*.

19449 ML Any memory locks established by the process via calls to *mlockall()* or *mlock()* shall be
19450 removed. If locked pages in the address space of the calling process are also mapped into
19451 the address spaces of other processes and are locked by those processes, the locks
19452 established by the other processes shall be unaffected by the call by this process to *_Exit()*
19453 or *_exit()*.

19454 Memory mappings that were created in the process shall be unmapped before the process
19455 is destroyed.

19456 TYM Any blocks of typed memory that were mapped in the calling process shall be unmapped,
19457 as if *munmap()* was implicitly called to unmap them.

19458 MSG All open message queue descriptors in the calling process shall be closed as if by
19459 appropriate calls to *mq_close()*.

19460 Any outstanding cancelable asynchronous I/O operations may be canceled. Those
19461 asynchronous I/O operations that are not canceled shall complete as if the *_Exit()* or
19462 *_exit()* operation had not yet occurred, but any associated signal notifications shall be
19463 suppressed. The *_Exit()* or *_exit()* operation may block awaiting such I/O completion.

19464 Whether any I/O is canceled, and which I/O may be canceled upon `_Exit()` or `_exit()`, is
19465 implementation-defined.

19466 Threads terminated by a call to `_Exit()` or `_exit()` shall not invoke their cancellation
19467 cleanup handlers or per-thread data destructors.

19468 OB TRC If the calling process is a trace controller process, any trace streams that were created by
19469 the calling process shall be shut down as described by the `posix_trace_shutdown()` function,
19470 and mapping of trace event names to trace event type identifiers of any process built for
19471 these trace streams may be deallocated.

19472 RETURN VALUE

19473 These functions do not return.

19474 ERRORS

19475 No errors are defined.

19476 EXAMPLES

19477 None.

19478 APPLICATION USAGE

19479 Normally applications should use `exit()` rather than `_Exit()` or `_exit()`.

19480 RATIONALE

19481 Process Termination

19482 Early proposals drew a distinction between normal and abnormal process termination.
19483 Abnormal termination was caused only by certain signals and resulted in implementation-
19484 defined “actions”, as discussed below. Subsequent proposals distinguished three types of
19485 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and
19486 *abnormal termination with actions*. Again the distinction between the two types of abnormal
19487 termination was that they were caused by different signals and that implementation-defined
19488 actions would result in the latter case. Given that these actions were completely implementation-
19489 defined, the early proposals were only saying when the actions could occur and how their
19490 occurrence could be detected, but not what they were. This was of little or no use to conforming
19491 applications, and thus the distinction is not made in this volume of POSIX.1-2017.

19492 The implementation-defined actions usually include, in most historical implementations, the
19493 creation of a file named **core** in the current working directory of the process. This file contains an
19494 image of the memory of the process, together with descriptive information about the process,
19495 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

19496 There is a potential security problem in creating a **core** file if the process was set-user-ID and the
19497 current user is not the owner of the program, if the process was set-group-ID and none of the
19498 user’s groups match the group of the program, or if the user does not have permission to write
19499 in the current directory. In this situation, an implementation either should not create a **core** file
19500 or should make it unreadable by the user.

19501 Despite the silence of this volume of POSIX.1-2017 on this feature, applications are advised not
19502 to create files named **core** because of potential conflicts in many implementations. Some
19503 implementations use a name other than **core** for the file; for example, by appending the process
19504 ID to the filename.

19505 **Terminating a Process**

19506 It is important that the consequences of process termination as described occur regardless of
19507 whether the process called `_exit()` (perhaps indirectly through `exit()`) or instead was terminated
19508 due to a signal or for some other reason. Note that in the specific case of `exit()` this means that
19509 the *status* argument to `exit()` is treated in the same way as the *status* argument to `_exit()`.

19510 A language other than C may have other termination primitives than the C-language `exit()`
19511 function, and programs written in such a language should use its native termination primitives,
19512 but those should have as part of their function the behavior of `_exit()` as described.
19513 Implementations in languages other than C are outside the scope of this version of this volume
19514 of POSIX.1-2017, however.

19515 As required by the ISO C standard, using **return** from `main()` has the same behavior (other than
19516 with respect to language scope issues) as calling `exit()` with the returned value. Reaching the end
19517 of the `main()` function has the same behavior as calling `exit(0)`.

19518 A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument *status*
19519 conventionally indicates successful termination. This corresponds to the specification for `exit()`
19520 in the ISO C standard. The convention is followed by utilities such as *make* and various shells,
19521 which interpret a zero status from a child process as success. For this reason, applications should
19522 not call `exit(0)` or `_exit(0)` when they terminate unsuccessfully; for example, in signal-catching
19523 functions.

19524 Historically, the implementation-defined process that inherits children whose parents have
19525 terminated without waiting on them is called *init* and has a process ID of 1.

19526 The sending of a `SIGHUP` to the foreground process group when a controlling process
19527 terminates corresponds to somewhat different historical implementations. In System V, the
19528 kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the
19529 kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is
19530 usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground
19531 process group with the *vhangup()* function. However, in 4.2 BSD, due to the behavior of the
19532 shells that support job control, the controlling process is usually a shell with no other processes
19533 in its process group. Thus, a change to make `_exit()` behave this way in such systems should not
19534 cause problems with existing applications.

19535 The termination of a process may cause a process group to become orphaned in either of two
19536 ways. The connection of a process group to its parent(s) outside of the group depends on both
19537 the parents and their children. Thus, a process group may be orphaned by the termination of the
19538 last connecting parent process outside of the group or by the termination of the last direct
19539 descendant of the parent process(es). In either case, if the termination of a process causes a
19540 process group to become orphaned, processes within the group are disconnected from their job
19541 control shell, which no longer has any information on the existence of the process group.
19542 Stopped processes within the group would languish forever. In order to avoid this problem,
19543 newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a
19544 `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP`
19545 signal causes the process group members to terminate unless they are catching or ignoring
19546 `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any
19547 of them are stopped.

19548 The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned
19549 process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each
19550 stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any
19551 additional descendants receive similar treatment at that time. In this volume of POSIX.1-2017,
19552 the signals are sent to the entire process group at the same time. Also, in this volume of

19553 POSIX.1-2017, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a
 19554 process group that is not orphaned; therefore, the action taken at `_exit()` must consider processes
 19555 other than child processes.

19556 It is possible for a process group to be orphaned by a call to `setpgid()` or `setsid()`, as well as by
 19557 process termination. This volume of POSIX.1-2017 does not require sending SIGHUP and
 19558 SIGCONT in those cases, because, unlike process termination, those cases are not caused
 19559 accidentally by applications that are unaware of job control. An implementation can choose to
 19560 send SIGHUP and SIGCONT in those cases as an extension; such an extension must be
 19561 documented as required in **<signal.h>**.

19562 The ISO/IEC 9899:1999 standard adds the `_Exit()` function that results in immediate program
 19563 termination without triggering signals or `atexit()`-registered functions. In POSIX.1-2017, this is
 19564 equivalent to the `_exit()` function.

19565 **FUTURE DIRECTIONS**

19566 None.

19567 **SEE ALSO**

19568 [*atexit\(\)*](#), [*exit\(\)*](#), [*mlock\(\)*](#), [*mlockall\(\)*](#), [*mq_close\(\)*](#), [*munmap\(\)*](#), [*posix_trace_create\(\)*](#), [*sem_close\(\)*](#),
 19569 [*semop\(\)*](#), [*setpgid\(\)*](#), [*setsid\(\)*](#), [*shmget\(\)*](#), [*wait\(\)*](#), [*waitid\(\)*](#)

19570 XBD **<stdlib.h>**, **<unistd.h>**

19571 **CHANGE HISTORY**

19572 First released in Issue 1. Derived from Issue 1 of the SVID.

19573 **Issue 5**

19574 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 19575 Threads Extension.

19576 Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

19577 The values of *status* from `exit()` are better described.

19578 **Issue 6**

19579 Extensions beyond the ISO C standard are marked.

19580 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics
 19581 for typed memory.

19582 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

19583 The `_Exit()` function is included.

19584 The DESCRIPTION is updated.

19585 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

19586 References to the `wait3()` function are removed.

19587 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the
 19588 DESCRIPTION.

19589 **Issue 7**

19590 Austin Group Interpretation 1003.1-2001 #031 is applied, separating these functions from the
 19591 `exit()` function.

19592 Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of
 19593 streams and closing of temporary files.

19594 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and

19595

Semaphores options is moved to the Base.

19596

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0033 [594] and XSH/TC2-2008/0034 [594,690] are applied.

19597

19598 **NAME**

19599 _longjmp, _setjmp ‡non-local goto

19600 **SYNOPSIS**

```
19601 OB XSI  #include <setjmp.h>
19602         void _longjmp(jmp_buf env, int val);
19603         int  _setjmp(jmp_buf env);
```

19604 **DESCRIPTION**

19605 The *_longjmp()* and *_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,
 19606 respectively, with the additional restriction that *_longjmp()* and *_setjmp()* shall not manipulate
 19607 the signal mask.

19608 If *_longjmp()* is called even though *env* was never initialized by a call to *_setjmp()*, or when the
 19609 last such call was in a function that has since returned, the results are undefined.

19610 **RETURN VALUE**19611 Refer to *longjmp()* and *setjmp()*.19612 **ERRORS**

19613 No errors are defined.

19614 **EXAMPLES**

19615 None.

19616 **APPLICATION USAGE**

19617 If *_longjmp()* is executed and the environment in which *_setjmp()* was executed no longer exists,
 19618 errors can occur. The conditions under which the environment of the *_setjmp()* no longer exists
 19619 include exiting the function that contains the *_setjmp()* call, and exiting an inner block with
 19620 temporary storage. This condition might not be detectable, in which case the *_longjmp()* occurs
 19621 and, if the environment no longer exists, the contents of the temporary storage of an inner block
 19622 are unpredictable. This condition might also cause unexpected process termination. If the
 19623 function has returned, the results are undefined.

19624 Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *_longjmp()* a pointer to a
 19625 buffer not created by *_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by
 19626 *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user
 19627 can cause all the problems listed above, and more.

19628 The *_longjmp()* and *_setjmp()* functions are included to support programs written to historical
 19629 system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

19630 **RATIONALE**

19631 None.

19632 **FUTURE DIRECTIONS**19633 The *_longjmp()* and *_setjmp()* functions may be removed in a future version.19634 **SEE ALSO**19635 *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*19636 XBD **<setjmp.h>**19637 **CHANGE HISTORY**

19638 First released in Issue 4, Version 2.

19639 **Issue 5**

19640 Moved from X/OPEN UNIX extension to BASE.

19641 **Issue 7**

19642 The *_longjmp()* and *_setjmp()* functions are marked obsolescent.

19643 **NAME**19644 `_tolower` — transliterate uppercase characters to lowercase19645 **SYNOPSIS**19646 OB XSI

```
#include <ctype.h>
19647 int _tolower(int c);
```

19648 **DESCRIPTION**19649 The `_tolower()` macro shall be equivalent to `tolower(c)` except that the application shall ensure
19650 that the argument `c` is an uppercase letter.19651 **RETURN VALUE**19652 Upon successful completion, `_tolower()` shall return the lowercase letter corresponding to the
19653 argument passed.19654 **ERRORS**

19655 No errors are defined.

19656 **EXAMPLES**

19657 None.

19658 **APPLICATION USAGE**19659 Applications should use the `tolower()` function instead of the obsolescent `_tolower()` function.19660 **RATIONALE**

19661 None.

19662 **FUTURE DIRECTIONS**19663 The `_tolower()` function may be removed in a future version.19664 **SEE ALSO**19665 [*`tolower\(\)`*](#), [*`isupper\(\)`*](#)19666 XBD [Chapter 7](#) (on page 135), [`<ctype.h>`](#)19667 **CHANGE HISTORY**

19668 First released in Issue 1. Derived from Issue 1 of the SVID.

19669 **Issue 6**

19670 The normative text is updated to avoid use of the term “must” for application requirements.

19671 **Issue 7**19672 The `_tolower()` function is marked obsolescent.

19673 **NAME**

19674 _toupper — transliterate lowercase characters to uppercase

19675 **SYNOPSIS**

```
19676 OB XSI #include <ctype.h>
19677        int _toupper(int c);
```

19678 **DESCRIPTION**

19679 The *_toupper()* macro shall be equivalent to *toupper()* except that the application shall ensure
19680 that the argument *c* is a lowercase letter.

19681 **RETURN VALUE**

19682 Upon successful completion, *_toupper()* shall return the uppercase letter corresponding to the
19683 argument passed.

19684 **ERRORS**

19685 No errors are defined.

19686 **EXAMPLES**

19687 None.

19688 **APPLICATION USAGE**

19689 Applications should use the *toupper()* function instead of the obsolescent *_toupper()* function.

19690 **RATIONALE**

19691 None.

19692 **FUTURE DIRECTIONS**

19693 The *_toupper()* function may be removed in a future version.

19694 **SEE ALSO**

19695 *islower()*, *toupper()*

19696 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#)

19697 **CHANGE HISTORY**

19698 First released in Issue 1. Derived from Issue 1 of the SVID.

19699 **Issue 6**

19700 The normative text is updated to avoid use of the term “must” for application requirements.

19701 **Issue 7**

19702 The *_toupper()* function is marked obsolescent.

19703 **NAME**

19704 a64l, l64a †convert between a 32-bit integer and a radix-64 ASCII string

19705 **SYNOPSIS**

```
19706 XSI #include <stdlib.h>
19707 long a64l(const char *s);
19708 char *l64a(long value);
```

19709 **DESCRIPTION**

19710 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by
 19711 which 32-bit integers can be represented by up to six characters; each character represents a digit
 19712 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall
 19713 be used for these operations.

19714 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for [2,11],
 19715 'A' through 'Z' for [12,37], and 'a' through 'z' for [38,63].

19716 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the
 19717 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains
 19718 more than six characters, *a64l()* shall use the first six. If the first six characters of the string
 19719 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The
 19720 *a64l()* function shall scan the character string from left to right with the least significant digit on
 19721 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than
 19722 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null
 19723 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

19724 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding
 19725 radix-64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

19726 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may
 19727 overwrite the buffer.

19728 The *l64a()* function need not be thread-safe.

19729 **RETURN VALUE**

19730 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the
 19731 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

19732 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall
 19733 return a pointer to an empty string.

19734 **ERRORS**

19735 No errors are defined.

19736 **EXAMPLES**

19737 None.

19738 **APPLICATION USAGE**

19739 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

19740 **RATIONALE**

19741 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

19742 **FUTURE DIRECTIONS**

19743 None.

19744 **SEE ALSO**19745 [strtol\(\)](#)19746 XBD [<stdlib.h>](#)19747 XCU [uuencode](#)19748 **CHANGE HISTORY**

19749 First released in Issue 4, Version 2.

19750 **Issue 5**

19751 Moved from X/OPEN UNIX extension to BASE.

19752 Normative text previously in the APPLICATION USAGE section is moved to the
19753 DESCRIPTION.19754 A note indicating that the *l64a()* function need not be reentrant is added to the DESCRIPTION.19755 **Issue 7**

19756 Austin Group Interpretation 1003.1-2001 #156 is applied.

19757 **NAME**

19758 abort — generate an abnormal process abort

19759 **SYNOPSIS**

19760 #include <stdlib.h>

19761 void abort(void);

19762 **DESCRIPTION**

19763 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19764 conflict between the requirements described here and the ISO C standard is unintentional. This
 19765 volume of POSIX.1-2017 defers to the ISO C standard.

19766 The *abort()* function shall cause abnormal process termination to occur, unless the signal
 19767 SIGABRT is being caught and the signal handler does not return.

19768 CX The abnormal termination processing shall include the default actions defined for SIGABRT and
 19769 may include an attempt to effect *fclose()* on all open streams.

19770 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the
 19771 argument SIGABRT.

19772 CX The status made available to *wait()*, *waitid()*, or *waitpid()* by *abort()* shall be that of a process
 19773 terminated by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the
 19774 SIGABRT signal.

19775 **RETURN VALUE**19776 The *abort()* function shall not return.19777 **ERRORS**

19778 No errors are defined.

19779 **EXAMPLES**

19780 None.

19781 **APPLICATION USAGE**

19782 Catching the signal is intended to provide the application developer with a portable means to
 19783 abort processing, free from possible interference from any implementation-supplied functions.

19784 **RATIONALE**

19785 The ISO/IEC 9899:1999 standard requires the *abort()* function to be async-signal-safe. Since
 19786 POSIX.1-2017 defers to the ISO C standard, this required a change to the DESCRIPTION from
 19787 “shall include the effect of *fclose()*” to “may include an attempt to effect *fclose()*.”

19788 The revised wording permits some backwards-compatibility and avoids a potential deadlock
 19789 situation.

19790 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded
 19791 paragraph from the DESCRIPTION:

19792 “On XSI-conformant systems, in addition the abnormal termination processing shall include the
 19793 effect of *fclose()* on message catalog descriptors.”

19794 There were several reasons to remove this paragraph:

19795 No special processing of open message catalogs needs to be performed prior to abnormal
 19796 process termination.

19797 The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open
 19798 streams is to flush output queued on the stream. Message catalogs in this context are read-
 19799 only and, therefore, do not need to be flushed.

19800 The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog
19801 descriptors are allowed, but not required to be implemented using a file descriptor, but
19802 there is no mention in POSIX.1-2017 of a message catalog descriptor using a standard I/O
19803 stream FILE object as would be expected by *fclose()*.

19804 FUTURE DIRECTIONS

19805 None.

19806 SEE ALSO

19807 *exit()*, *kill()*, *raise()*, *signal()*, *wait()*, *waitid()*

19808 XBD <[stdlib.h](#)>

19809 CHANGE HISTORY

19810 First released in Issue 1. Derived from Issue 1 of the SVID.

19811 Issue 6

19812 Extensions beyond the ISO C standard are marked.

19813 Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

19814 The Open Group Base Resolution bwg2002-003 is applied.

19815 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the
19816 DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

19817 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing “implementation-
19818 defined functions” to “implementation-supplied functions” in the APPLICATION USAGE
19819 section.

19820 **NAME**

19821 abs — return an integer absolute value

19822 **SYNOPSIS**

19823 #include <stdlib.h>

19824 int abs(int i);

19825 **DESCRIPTION**

19826 CX The functionality described on this reference page is aligned with the ISO C standard. Any
19827 conflict between the requirements described here and the ISO C standard is unintentional. This
19828 volume of POSIX.1-2017 defers to the ISO C standard.

19829 The *abs()* function shall compute the absolute value of its integer operand, *i*. If the result cannot
19830 be represented, the behavior is undefined.

19831 **RETURN VALUE**

19832 The *abs()* function shall return the absolute value of its integer operand.

19833 **ERRORS**

19834 No errors are defined.

19835 **EXAMPLES**

19836 None.

19837 **APPLICATION USAGE**

19838 In two's-complement representation, the absolute value of the negative integer with largest
19839 magnitude {INT_MIN} might not be representable.

19840 **RATIONALE**

19841 None.

19842 **FUTURE DIRECTIONS**

19843 None.

19844 **SEE ALSO**

19845 *fabs()*, *labs()*

19846 XBD <stdlib.h>

19847 **CHANGE HISTORY**

19848 First released in Issue 1. Derived from Issue 1 of the SVID.

19849 **Issue 6**

19850 Extensions beyond the ISO C standard are marked.

19851 **NAME**

19852 accept ‡'accept a new connection on a socket

19853 **SYNOPSIS**

```
19854       #include <sys/socket.h>
19855       int accept(int socket, struct sockaddr *restrict address,
19856                 socklen_t *restrict address_len);
```

19857 **DESCRIPTION**

19858 The *accept()* function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket. The file descriptor shall be allocated as described in [Section 2.14](#) (on page 549).

19862 The *accept()* function takes the following arguments:

19863 *socket* Specifies a socket that was created with *socket()*, has been bound to an address with *bind()*, and has issued a successful call to *listen()*.

19865 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of the connecting socket shall be returned.

19867 *address_len* Either a null pointer, if *address* is a null pointer, or a pointer to a **socklen_t** object which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address.

19870 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in the object pointed to by *address_len*.

19873 If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

19875 If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

19877 If the listen queue is empty of connection requests and **O_NONBLOCK** is not set on the file descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is empty of connection requests and **O_NONBLOCK** is set on the file descriptor for the socket, *accept()* shall fail and set *errno* to **[EAGAIN]** or **[EWOULDBLOCK]**.

19881 The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

19883 **RETURN VALUE**

19884 Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted socket. Otherwise, **-1** shall be returned, *errno* shall be set to indicate the error, and any object pointed to by *address_len* shall remain unchanged.

19887 **ERRORS**

19888 The *accept()* function shall fail if:

19889 **[EAGAIN]** or **[EWOULDBLOCK]**

19890 **O_NONBLOCK** is set for the socket file descriptor and no connections are present to be accepted.

19892 **[EBADF]** The *socket* argument is not a valid file descriptor.

19893		[ECONNABORTED]	
19894			A connection has been aborted.
19895		[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
19896			
19897		[EINVAL]	The <i>socket</i> is not accepting connections.
19898		[EMFILE]	All file descriptors available to the process are currently open.
19899		[ENFILE]	The maximum number of file descriptors in the system are already open.
19900		[ENOBUFS]	No buffer space is available.
19901		[ENOMEM]	There was insufficient memory available to complete the operation.
19902		[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
19903		[EOPNOTSUPP]	The socket type of the specified socket does not support accepting connections.
19904			
19905			The <i>accept()</i> function may fail if:
19906	OB XSR	[EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
19907			

EXAMPLES

19908 None.

APPLICATION USAGE

19911 When a connection is available, *select()* indicates that the file descriptor for the socket is ready for reading.

RATIONALE

19914 None.

FUTURE DIRECTIONS

19916 None.

SEE ALSO

19918 [Section 2.14](#) (on page 549), *bind()*, *connect()*, *listen()*, *socket()*

19919 XBD [<sys/socket.h>](#)

CHANGE HISTORY

19921 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

19922 The **restrict** keyword is added to the *accept()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

19925 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

19926 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS] and [ENOMEM] errors to become “shall fail” errors.

19928 Functionality relating to XSI STREAMS is marked obsolescent.

19929 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0018 [464] is applied.

19930 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0035 [835] and XSH/TC2-2008/0036 [836] are applied.

19932 **NAME**

19933 access, faccessat †determine accessibility of a file descriptor

19934 **SYNOPSIS**

19935 #include <unistd.h>

19936 int access(const char *path, int amode);

19937 OH #include <fcntl.h>

19938 int faccessat(int fd, const char *path, int amode, int flag);

19939 **DESCRIPTION**

19940 The *access()* function shall check the file named by the pathname pointed to by the *path*
 19941 argument for accessibility according to the bit pattern contained in *amode*. The checks for
 19942 accessibility (including directory permissions checked during pathname resolution) shall be
 19943 performed using the real user ID in place of the effective user ID and the real group ID in place
 19944 of the effective group ID.

19945 The value of *amode* is either the bitwise-inclusive OR of the access permissions to be checked
 19946 (R_OK, W_OK, X_OK) or the existence test (F_OK).

19947 If any access permissions are checked, each shall be checked individually, as described in XBD
 19948 Section 4.5 (on page 108), except that where that description refers to execute permission for a
 19949 process with appropriate privileges, an implementation may indicate success for X_OK even if
 19950 execute permission is not granted to any user.

19951 The *faccessat()* function, when called with a *flag* value of zero, shall be equivalent to the *access()*
 19952 function, except in the case where *path* specifies a relative path. In this case the file whose
 19953 accessibility is to be determined shall be located relative to the directory associated with the file
 19954 descriptor *fd* instead of the current working directory. If the access mode of the open file
 19955 description associated with the file descriptor is not O_SEARCH, the function shall check
 19956 whether directory searches are permitted using the current permissions of the directory
 19957 underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform
 19958 the check.

19959 If *faccessat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 19960 directory shall be used and, if *flag* is zero, the behavior shall be identical to a call to *access()*.

19961 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 19962 in <fcntl.h>:

19963 AT_EACCESS The checks for accessibility (including directory permissions checked during
 19964 pathname resolution) shall be performed using the effective user ID and
 19965 group ID instead of the real user ID and group ID as required in a call to
 19966 *access()*.

19967 **RETURN VALUE**

19968 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 19969 return -1 and set *errno* to indicate the error.

19970 **ERRORS**

19971 These functions shall fail if:

19972 [EACCES] Permission bits of the file mode do not permit the requested access, or search
 19973 permission is denied on a component of the path prefix.

19974 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 19975 argument.

19976	[ENAMETOOLONG]	
19977		The length of a component of a pathname is longer than {NAME_MAX}.
19978	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
19979	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>path</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
19980		
19981		
19982		
19983		
19984	[EROFS]	Write access is requested for a file on a read-only file system.
19985		The <i>faccessat()</i> function shall fail if:
19986	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not <code>O_SEARCH</code> and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
19987		
19988		
19989	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
19990		
19991	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
19992		
19993		These functions may fail if:
19994	[EINVAL]	The value of the <i>amode</i> argument is invalid.
19995	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
19996		
19997	[ENAMETOOLONG]	
19998		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
19999		
20000		
20001	[ETXTBSY]	Write access is requested for a pure procedure (shared text) file that is being executed.
20002		
20003		The <i>faccessat()</i> function may fail if:
20004	[EINVAL]	The value of the <i>flag</i> argument is not valid.

20005 EXAMPLES

20006 Testing for the Existence of a File

20007 The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```

20008 #include <unistd.h>
20009 ...
20010 int result;
20011 const char *pathname = "/tmp/myfile";
20012 result = access (pathname, F_OK);

```

20013 **APPLICATION USAGE**

20014 Use of these functions is discouraged since by the time the returned information is acted upon, it
 20015 is out-of-date. (That is, acting upon the information always leads to a time-of-check-to-time-of-
 20016 use race condition.) An application should instead attempt the action itself and handle the
 20017 [EACCES] error that occurs if the file is not accessible (with a change of effective user and group
 20018 IDs beforehand, and perhaps a change back afterwards, in the case where *access()* or *faccessat()*
 20019 without AT_EACCESS would have been used.)

20020 Historically, one of the uses of *access()* was in set-user-ID root programs to check whether the
 20021 user running the program had access to a file. This relied on “super-user” privileges which were
 20022 granted based on the effective user ID being zero, so that when *access()* used the real user ID to
 20023 check accessibility those privileges were not taken into account. On newer systems where
 20024 privileges can be assigned which have no association with user or group IDs, if a program with
 20025 such privileges calls *access()*, the change of IDs has no effect on the privileges and therefore they
 20026 are taken into account in the accessibility checks. Thus, *access()* (and *faccessat()* with flag zero)
 20027 cannot be used for this historical purpose in such programs. Likewise, if a system provides any
 20028 additional or alternate file access control mechanisms that are not user ID-based, they will still
 20029 be taken into account.

20030 If a relative pathname is used, no account is taken of whether the current directory (or the
 20031 directory associated with the file descriptor *fd*) is accessible via any absolute pathname.
 20032 Applications using *access()*, or *faccessat()* without AT_EACCESS, may consequently act as if the
 20033 file would be accessible to a user with the real user ID and group ID of the process when such a
 20034 user would not in practice be able to access the file because access would be denied at some
 20035 point above the current directory (or the directory associated with the file descriptor *fd*) in the
 20036 file hierarchy.

20037 If *access()* or *faccessat()* is used with W_OK to check for write access to a directory which has the
 20038 S_ISVTX bit set, a return value indicating the directory is writable can be misleading since some
 20039 operations on files in the directory would not be permitted based on the ownership of those files
 20040 (see XBD Section 4.3, on page 108).

20041 Additional values of *amode* other than the set defined in the description may be valid; for
 20042 example, if a system has extended access controls.

20043 The use of the AT_EACCESS value for *flag* enables functionality not available in *access()*.

20044 **RATIONALE**

20045 In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()*
 20046 function because:

- 20047 1. Historical implementations of *access()* do not test file access correctly when the process’
 20048 real user ID is superuser. In particular, they always return zero when testing execute
 20049 permissions without regard to whether the file is executable.
- 20050 2. The superuser has complete access to all files on a system. As a consequence, programs
 20051 started by the superuser and switched to the effective user ID with lesser privileges
 20052 cannot use *access()* to test their file access permissions.

20053 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of
 20054 POSIX.1-2017 now allows *access()* to behave in the desired way because several implementations
 20055 have corrected the problem. It was also argued that problem (2) is more easily solved by using
 20056 *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the error, rather
 20057 than creating a new function that would not be as reliable. Therefore, *eaccess()* is not included in
 20058 this volume of POSIX.1-2017.

20059 The sentence concerning appropriate privileges and execute permission bits reflects the two

20060 possibilities implemented by historical implementations when checking superuser access for
20061 X_OK.

20062 New implementations are discouraged from returning X_OK unless at least one execution
20063 permission bit is set.

20064 The purpose of the *faccessat()* function is to enable the checking of the accessibility of files in
20065 directories other than the current working directory without exposure to race conditions. Any
20066 part of the path of a file could be changed in parallel to a call to *access()*, resulting in unspecified
20067 behavior. By opening a file descriptor for the target directory and using the *faccessat()* function it
20068 can be guaranteed that the file tested for accessibility is located relative to the desired directory.

20069 **FUTURE DIRECTIONS**

20070 These functions may be formally deprecated (for example, by shading them OB) in a future
20071 version of this standard.

20072 **SEE ALSO**

20073 *chmod()*, *fstatat()*

20074 XBD Section 4.5 (on page 108), [<fcntl.h>](#), [<unistd.h>](#)

20075 **CHANGE HISTORY**

20076 First released in Issue 1. Derived from Issue 1 of the SVID.

20077 **Issue 6**

20078 The following new requirements on POSIX implementations derive from alignment with the
20079 Single UNIX Specification:

20080 The [ELOOP] mandatory error condition is added.

20081 A second [ENAMETOOLONG] is added as an optional error condition.

20082 The [ETXTBSY] optional error condition is added.

20083 The following changes were made to align with the IEEE P1003.1a draft standard:

20084 The [ELOOP] optional error condition is added.

20085 **Issue 7**

20086 Austin Group Interpretations 1003.1-2001 #046 and #143 are applied.

20087 The *faccessat()* function is added from The Open Group Technical Standard, 2006, Extended API
20088 Set Part 2.

20089 Changes are made to allow a directory to be opened for searching.

20090 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
20091 pathname exists but is not a directory or a symbolic link to a directory.

20092 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0019 [461], XSH/TC1-2008/0020 [324],
20093 XSH/TC1-2008/0021 [278], XSH/TC1-2008/0022 [278], and XSH/TC1-2008/0023 [291] are
20094 applied.

20095 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0037 [873], XSH/TC2-2008/0038 [591],
20096 XSH/TC2-2008/0039 [838], XSH/TC2-2008/0040 [817], XSH/TC2-2008/0041 [487],
20097 XSH/TC2-2008/0042 [838], XSH/TC2-2008/0043 [817], and XSH/TC2-2008/0044 [838] are
20098 applied.

20099 **NAME**20100 acos, acosf, acosl \dagger arc cosine functions20101 **SYNOPSIS**

```
20102       #include <math.h>
20103       double acos(double x);
20104       float acosf(float x);
20105       long double acosl(long double x);
```

20106 **DESCRIPTION**

20107 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 20108 conflict between the requirements described here and the ISO C standard is unintentional. This
 20109 volume of POSIX.1-2017 defers to the ISO C standard.

20110 These functions shall compute the principal value of the arc cosine of their argument x . The
 20111 value of x should be in the range $[-1,1]$.

20112 An application wishing to check for error situations should set *errno* to zero and call
 20113 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20114 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20115 zero, an error has occurred.

20116 **RETURN VALUE**

20117 Upon successful completion, these functions shall return the arc cosine of x , in the range $[0,\pi]$
 20118 radians.

20119 **MX** For finite values of x not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 20120 supported), or an implementation-defined value shall be returned.

20121 **MX** If x is NaN, a NaN shall be returned.

20122 If x is +1, +0 shall be returned.

20123 If x is $\pm\text{Inf}$, a domain error shall occur, and a NaN shall be returned.

20124 **ERRORS**

20125 These functions shall fail if:

20126 **MX** Domain Error The x argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.

20127 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20128 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 20129 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 20130 shall be raised.

20131 **EXAMPLES**

20132 None.

20133 **APPLICATION USAGE**

20134 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 20135 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20136 **RATIONALE**

20137 None.

20138 **FUTURE DIRECTIONS**

20139 None.

20140 **SEE ALSO**20141 *cos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*20142 XBD Section 4.20 (on page 117), **<math.h>**20143 **CHANGE HISTORY**

20144 First released in Issue 1. Derived from Issue 1 of the SVID.

20145 **Issue 5**20146 The DESCRIPTION is updated to indicate how an application should check for an error. This
20147 text was previously published in the APPLICATION USAGE section.20148 **Issue 6**20149 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.20150 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
20151 revised to align with the ISO/IEC 9899:1999 standard.20152 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
20153 marked.20154 **Issue 7**

20155 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0024 [320] is applied.

20156 **NAME**

20157 acosh, acoshf, acoshl ‡inverse hyperbolic cosine functions

20158 **SYNOPSIS**

```
20159 #include <math.h>
20160 double acosh(double x);
20161 float acoshf(float x);
20162 long double acoshl(long double x);
```

20163 **DESCRIPTION**

20164 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20165 conflict between the requirements described here and the ISO C standard is unintentional. This
 20166 volume of POSIX.1-2017 defers to the ISO C standard.

20167 These functions shall compute the inverse hyperbolic cosine of their argument x .

20168 An application wishing to check for error situations should set *errno* to zero and call
 20169 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20170 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20171 zero, an error has occurred.

20172 **RETURN VALUE**

20173 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their
 20174 argument.

20175 MX For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an
 20176 implementation-defined value shall be returned.

20177 MX If x is NaN, a NaN shall be returned.

20178 If x is +1, +0 shall be returned.

20179 If x is +Inf, +Inf shall be returned.

20180 If x is -Inf, a domain error shall occur, and a NaN shall be returned.

20181 **ERRORS**

20182 These functions shall fail if:

20183 MX Domain Error The x argument is finite and less than +1.0, or is -Inf.

20184 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20185 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* &
 20186 MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 20187 shall be raised.

20188 **EXAMPLES**

20189 None.

20190 **APPLICATION USAGE**

20191 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 20192 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20193 **RATIONALE**

20194 None.

20195 **FUTURE DIRECTIONS**

20196 None.

20197 **SEE ALSO**20198 *cosh()*, *feclearexcept()*, *fetestexcept()*20199 XBD Section 4.20 (on page 117), **<math.h>**20200 **CHANGE HISTORY**

20201 First released in Issue 4, Version 2.

20202 **Issue 5**

20203 Moved from X/OPEN UNIX extension to BASE.

20204 **Issue 6**20205 The *acosh()* function is no longer marked as an extension.20206 The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999
20207 standard.20208 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
20209 revised to align with the ISO/IEC 9899:1999 standard.20210 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
20211 marked.20212 **Issue 7**

20213 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0025 [320] is applied.

20214 **NAME**

20215 acosl inverse cosine functions

20216 **SYNOPSIS**

20217 #include <math.h>

20218 long double acosl(long double x);

20219 **DESCRIPTION**20220 Refer to *acos()*.

20221 **NAME**

20222 aio_cancel — cancel an asynchronous I/O request

20223 **SYNOPSIS**

20224 #include <aio.h>

20225 int aio_cancel(int *fildev*, struct aiocb **aiocbp*);20226 **DESCRIPTION**

20227 The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests
 20228 currently outstanding against file descriptor *fildev*. The *aiocbp* argument points to the
 20229 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
 20230 all outstanding cancelable asynchronous I/O requests against *fildev* shall be canceled.

20231 Normal asynchronous notification shall occur for asynchronous I/O operations that are
 20232 successfully canceled. If there are requests that cannot be canceled, then the normal
 20233 asynchronous completion process shall take place for those requests when they are completed.

20234 For requested operations that are successfully canceled, the associated error status shall be set to
 20235 [ECANCELED] and the return status shall be -1. For requested operations that are not
 20236 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

20237 If *aiocbp* is not NULL, then if *fildev* does not have the same value as the file descriptor with which
 20238 the asynchronous operation was initiated, unspecified results occur.

20239 Which operations are cancelable is implementation-defined.

20240 **RETURN VALUE**

20241 The *aio_cancel()* function shall return the value AIO_CANCELED if the requested operation(s)
 20242 were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the
 20243 requested operation(s) cannot be canceled because it is in progress. In this case, the state of the
 20244 other operations, if any, referenced in the call to *aio_cancel()* is not indicated by the return value
 20245 of *aio_cancel()*. The application may determine the state of affairs for these operations by using
 20246 *aio_error()*. The value AIO_ALLDONE is returned if all of the operations have already
 20247 completed. Otherwise, the function shall return -1 and set *errno* to indicate the error.

20248 **ERRORS**

20249 The *aio_cancel()* function shall fail if:

20250 [EBADF] The *fildev* argument is not a valid file descriptor.

20251 **EXAMPLES**

20252 None.

20253 **APPLICATION USAGE**

20254 None.

20255 **RATIONALE**

20256 None.

20257 **FUTURE DIRECTIONS**

20258 None.

20259 **SEE ALSO**

20260 [aio_read\(\)](#), [aio_write\(\)](#)

20261 XBD [<aio.h>](#)

20262 **CHANGE HISTORY**

20263 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20264 **Issue 6**

20265 The [ENOSYS] error condition has been removed as stubs need not be provided if an
20266 implementation does not support the Asynchronous Input and Output option.

20267 The APPLICATION USAGE section is added.

20268 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words ``to the
20269 calling process'' in the RETURN VALUE section. The term was unnecessary and precluded
20270 threads.

20271 **Issue 7**

20272 The *aio_cancel()* function is moved from the Asynchronous Input and Output option to the Base.

20273 **NAME**

20274 aio_error — retrieve errors status for an asynchronous I/O operation

20275 **SYNOPSIS**

20276 #include <aio.h>

20277 int aio_error(const struct aiocb *aiocbp);

20278 **DESCRIPTION**

20279 The *aio_error()* function shall return the error status associated with the **aiocb** structure
 20280 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the
 20281 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*
 20282 operation. If the operation has not yet completed, then the error status shall be equal to
 20283 [EINPROGRESS].

20284 If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been
 20285 scheduled, the results are undefined.

20286 **RETURN VALUE**

20287 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
 20288 asynchronous operation has completed unsuccessfully, then the error status, as described for
 20289 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has
 20290 not yet completed, then [EINPROGRESS] shall be returned.

20291 If the *aio_error()* function fails, it shall return -1 and set *errno* to indicate the error.

20292 **ERRORS**

20293 The *aio_error()* function may fail if:

20294 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 20295 return status has not yet been retrieved.

20296 **EXAMPLES**

20297 None.

20298 **APPLICATION USAGE**

20299 None.

20300 **RATIONALE**

20301 None.

20302 **FUTURE DIRECTIONS**

20303 None.

20304 **SEE ALSO**

20305 *aio_cancel()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 20306 *lseek()*, *read()*

20307 XBD <aio.h>

20308 **CHANGE HISTORY**

20309 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20310 **Issue 6**

20311 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 20312 implementation does not support the Asynchronous Input and Output option.

20313 The APPLICATION USAGE section is added.

20314 **Issue 7**

20315 Austin Group Interpretation 1003.1-2001 #045 is applied.

20316 SD5-XSH-ERN-148 is applied.

20317 The *aio_error()* function is moved from the Asynchronous Input and Output option to the Base.

20318 **NAME**

20319 aio_fsync — asynchronous file synchronization

20320 **SYNOPSIS**

```
20321 FSC|SIO #include <aio.h>
20322 int aio_fsync(int op, struct aiocb *aiocbp);
```

20323 **DESCRIPTION**

20324 The *aio_fsync()* function shall asynchronously perform a file synchronization operation, as
 20325 specified by the *op* argument, for I/O operations associated with the file indicated by the file
 20326 descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and
 20327 queued at the time of the call to *aio_fsync()*. The function call shall return when the
 20328 synchronization request has been initiated or queued to the file or device (even when the data
 20329 cannot be synchronized immediately).

20330 SIO If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to
 20331 *fdatsync()*; that is, as defined for synchronized I/O data integrity completion.

20332 FSC If *op* is O_SYNC, all currently queued I/O operations shall be completed as if by a call to *fsync()*;
 20333 FSC SIO that is, as defined for synchronized I/O file integrity completion. If the *aio_fsync()* function
 20334 fails, or if the operation queued by *aio_fsync()* fails, then outstanding I/O operations are not
 20335 guaranteed to have been completed.

20336 If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to
 20337 *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of
 20338 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized
 20339 fashion.

20340 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used
 20341 as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return
 20342 status, respectively, of the asynchronous operation while it is proceeding. When the request is
 20343 queued, the error status for the operation is [EINPROGRESS]. When all data has been
 20344 successfully transferred, the error status shall be reset to reflect the success or failure of the
 20345 operation. If the operation does not complete successfully, the error status for the operation shall
 20346 be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to
 20347 occur as specified in Section 2.4.1 (on page 488) when all operations have achieved synchronized
 20348 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the
 20349 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O
 20350 completion, then the behavior is undefined.

20351 If the *aio_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to
 20352 have been successfully transferred.

20353 **RETURN VALUE**

20354 The *aio_fsync()* function shall return the value 0 if the I/O operation is successfully queued;
 20355 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

20356 **ERRORS**

20357 The *aio_fsync()* function shall fail if:

20358 [EAGAIN] The requested asynchronous operation was not queued due to temporary
 20359 resource limitations.

20360 [EBADF] The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument
 20361 is not a valid file descriptor.

- 20362 SIO [EINVAL] This implementation does not support synchronized I/O for this file.
- 20363 FSC [EINVAL] The *aio_fildes* member of the **aio_cb** structure refers to a file on which an *fsync()*
20364 operation is not possible.
- 20365 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified, or O_DSYNC
20366 was specified and the implementation does not provide runtime support for
20367 the Synchronized Input and Output option, or O_SYNC was specified and the
20368 implementation does not provide runtime support for the File
20369 Synchronization option.
- 20370 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error
20371 condition defined for *read()* and *write()*. The error is returned in the error status for the
20372 asynchronous operation, which can be retrieved using *aio_error()*.

EXAMPLES

20373 None.
20374

APPLICATION USAGE

20375 Note that even if the file descriptor is not open for writing, if there are any pending write
20376 requests on the underlying file, then that I/O will be completed prior to the return of a call to
20377 *aio_error()* or *aio_return()* indicating that the operation has completed.
20378

RATIONALE

20379 None.
20380

FUTURE DIRECTIONS

20381 None.
20382

SEE ALSO

20383 *aio_error()*, *aio_return()*, *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read()*, *write()*
20384

20385 XBD <[aio.h](#)>

CHANGE HISTORY

20386 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
20387

Issue 6

20388 The [ENOSYS] error condition has been removed as stubs need not be provided if an
20389 implementation does not support the Asynchronous Input and Output option.
20390

20391 The APPLICATION USAGE section is added.

20392 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words “to the
20393 calling process” in the RETURN VALUE section. The term was unnecessary and precluded
20394 threads.

Issue 7

20395 The *aio_fsync()* function is moved from the Asynchronous Input and Output option to the Base.
20396

20397 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0026 [98] and XSH/TC1-2008/0027
20398 [98] are applied.

20399 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0045 [671] is applied.

20400 **NAME**

20401 aio_read — asynchronous read from a file

20402 **SYNOPSIS**

20403 #include <aio.h>

20404 int aio_read(struct aiocb *aiocbp);

20405 **DESCRIPTION**

20406 The *aio_read()* function shall read *aiocbp* ~~*aio_nbytes*~~ from the file associated with
 20407 *aiocbp* ~~*aio_fildes*~~ into the buffer pointed to by *aiocbp* ~~*aio_buf*~~. The function call shall return
 20408 when the read request has been initiated or queued to the file or device (even when the data
 20409 cannot be delivered immediately).

20410 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 20411 at a priority equal to a base scheduling priority minus *aiocbp* ~~*aio_reqprio*~~. If Thread Execution
 20412 Scheduling is not supported, then the base scheduling priority is that of the calling process;
 20413 PIO TPS otherwise, the base scheduling priority is that of the calling thread.

20414 The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to
 20415 determine the error status and return status, respectively, of the asynchronous operation while it
 20416 is proceeding. If an error condition is encountered during queuing, the function call shall return
 20417 without having initiated or queued the request. The requested operation takes place at the
 20418 absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to
 20419 the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. After a
 20420 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
 20421 is unspecified.

20422 The *aio_sigevent* member specifies the notification which occurs when the request is completed.

20423 The *aiocbp* ~~*aio_lio_opcode*~~ field shall be ignored by *aio_read()*.

20424 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp* ~~*aio_buf*~~ or
 20425 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 20426 completion, then the behavior is undefined.

20427 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

20428 SIO If synchronized I/O is enabled on the file associated with *aiocbp* ~~*aio_fildes*~~, the behavior of this
 20429 function shall be according to the definitions of synchronized I/O data integrity completion and
 20430 synchronized I/O file integrity completion.

20431 For any system action that changes the process memory space while an asynchronous I/O is
 20432 outstanding to the address range being changed, the result of that action is undefined.

20433 For regular files, no data transfer shall occur past the offset maximum established in the open
 20434 file description associated with *aiocbp* ~~*aio_fildes*~~.

20435 **RETURN VALUE**

20436 The *aio_read()* function shall return the value zero if the I/O operation is successfully queued;
 20437 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

20438 **ERRORS**

20439 The *aio_read()* function shall fail if:

20440 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 20441 resource limitations.

20442 Each of the following conditions may be detected synchronously at the time of the call to
 20443 *aio_read()*, or asynchronously. If any of the conditions below are detected synchronously, the

20444 *aio_read()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 20445 conditions below are detected asynchronously, the return status of the asynchronous operation
 20446 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.

20447 [EBADF] The *aio* *fd* argument is not a valid file descriptor open for reading.

20448 [EINVAL] The file offset value implied by *aio* *offset* would be invalid,
 20449 *aio* *reqprio* is not a valid value, or *aio* *nbytes* is an invalid
 20450 value.

20451 In the case that the *aio_read()* successfully queues the I/O operation but the operation is
 20452 subsequently canceled or encounters an error, the return status of the asynchronous operation is
 20453 one of the values normally returned by the *read()* function call. In addition, the error status of
 20454 the asynchronous operation is set to one of the error statuses normally set by the *read()* function
 20455 call, or one of the following values:

20456 [EBADF] The *aio* *fd* argument is not a valid file descriptor open for reading.

20457 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 20458 *aio_cancel()* request.

20459 [EINVAL] The file offset value implied by *aio* *offset* would be invalid.

20460 The following condition may be detected synchronously or asynchronously:

20461 [EOVERFLOW] The file is a regular file, *aio* *nbytes* is greater than 0, and the starting
 20462 offset in *aio* *offset* is before the end-of-file and is at or beyond the
 20463 offset maximum in the open file description associated with *aio* *fd*.

20464 EXAMPLES

20465 None.

20466 APPLICATION USAGE

20467 None.

20468 RATIONALE

20469 None.

20470 FUTURE DIRECTIONS

20471 None.

20472 SEE ALSO

20473 [aio_cancel\(\)](#), [aio_error\(\)](#), [lio_listio\(\)](#), [aio_return\(\)](#), [aio_write\(\)](#), [close\(\)](#), [exec](#), [exit\(\)](#), [fork\(\)](#), [lseek\(\)](#),
 20474 [read\(\)](#)

20475 XBD [<aio.h>](#)

20476 CHANGE HISTORY

20477 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20478 Large File Summit extensions are added.

20479 Issue 6

20480 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 20481 implementation does not support the Asynchronous Input and Output option.

20482 The APPLICATION USAGE section is added.

20483 The following new requirements on POSIX implementations derive from alignment with the
20484 Single UNIX Specification:

20485 In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open
20486 file description. This change is to support large files.

20487 In the ERRORS section, the [Eoverflow] condition is added. This change is to support
20488 large files.

20489 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the
20490 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the
20491 words "to the calling process" in the RETURN VALUE section.

20492 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL]
20493 error, so that detection of an [EINVAL] error for an invalid value of *aiocbp* ~~*aio_reqprio*~~ is only
20494 required if the Prioritized Input and Output option is supported.

20495 **Issue 7**

20496 Austin Group Interpretation 1003.1-2001 #082 is applied.

20497 The *aio_read()* function is moved from the Asynchronous Input and Output option to the Base.

20498 **NAME**

20499 aio_return — retrieve return status of an asynchronous I/O operation

20500 **SYNOPSIS**

20501 #include <aio.h>

20502 ssize_t aio_return(struct aiocb *aiocbp);

20503 **DESCRIPTION**

20504 The *aio_return()* function shall return the return status associated with the **aiocb** structure
 20505 referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the
 20506 value that would be returned by the corresponding *read()*, *write()*, or *fsync()* function call. If the
 20507 error status for the operation is equal to [EINPROGRESS], then the return status for the
 20508 operation is undefined. The *aio_return()* function may be called exactly once to retrieve the
 20509 return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in
 20510 a call to *aio_return()* or *aio_error()*, an error may be returned. When the **aiocb** structure referred
 20511 to by *aiocbp* is used to submit another asynchronous operation, then *aio_return()* may be
 20512 successfully used to retrieve the return status of that operation.

20513 **RETURN VALUE**

20514 If the asynchronous I/O operation has completed, then the return status, as described for *read()*,
 20515 *write()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has not yet completed,
 20516 the results of *aio_return()* are undefined.

20517 If the *aio_return()* function fails, it shall return `-1` and set *errno* to indicate the error.

20518 **ERRORS**20519 The *aio_return()* function may fail if:

20520 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 20521 return status has not yet been retrieved.

20522 **EXAMPLES**

20523 None.

20524 **APPLICATION USAGE**

20525 None.

20526 **RATIONALE**

20527 None.

20528 **FUTURE DIRECTIONS**

20529 None.

20530 **SEE ALSO**

20531 [aio_cancel\(\)](#), [aio_error\(\)](#), [aio_fsync\(\)](#), [aio_read\(\)](#), [aio_write\(\)](#), [close\(\)](#), [exec](#), [exit\(\)](#), [fork\(\)](#), [lio_listio\(\)](#),
 20532 [lseek\(\)](#), [read\(\)](#)

20533 XBD [<aio.h>](#)20534 **CHANGE HISTORY**

20535 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20536 **Issue 6**

20537 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 20538 implementation does not support the Asynchronous Input and Output option.

20539 The APPLICATION USAGE section is added.

20540 The [EINVAL] error condition is made optional. This is for consistency with the DESCRIPTION.

20541 **Issue 7**

20542 SD5-XSH-ERN-148 is applied.

20543 The *aio_return()* function is moved from the Asynchronous Input and Output option to the Base.

20544 **NAME**

20545 aio_suspend — wait for an asynchronous I/O request

20546 **SYNOPSIS**

```
20547 #include <aio.h>
20548 int aio_suspend(const struct aiocb *const list[], int nent,
20549               const struct timespec *timeout);
```

20550 **DESCRIPTION**

20551 The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous
 20552 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
 20553 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
 20554 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
 20555 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
 20556 function shall return without suspending the calling thread. The *list* argument is an array of
 20557 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
 20558 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
 20559 asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain null
 20560 pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have
 20561 not been used in submitting asynchronous I/O, the effect is undefined.

20562 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
 20563 the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an error.

20564 MON If the Monotonic Clock option is supported, the clock that shall be used to measure this time
 20565 interval shall be the CLOCK_MONOTONIC clock.

20566 **RETURN VALUE**

20567 If the *aio_suspend()* function returns after one or more asynchronous I/O operations have
 20568 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and
 20569 set *errno* to indicate the error.

20570 The application may determine which asynchronous I/O completed by scanning the associated
 20571 error and return status using *aio_error()* and *aio_return()*, respectively.

20572 **ERRORS**

20573 The *aio_suspend()* function shall fail if:

20574 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
 20575 time interval indicated by *timeout*.

20576 [EINTR] A signal interrupted the *aio_suspend()* function. Note that, since each
 20577 asynchronous I/O operation may possibly provoke a signal when it
 20578 completes, this error return may be caused by the completion of one (or more)
 20579 of the very I/O operations being awaited.

20580 **EXAMPLES**

20581 None.

20582 **APPLICATION USAGE**

20583 None.

20584 **RATIONALE**

20585 None.

20586 **FUTURE DIRECTIONS**

20587 None.

20588 **SEE ALSO**20589 *aio_read()*, *aio_write()*, *lio_listio()*20590 XBD <**aio.h**>20591 **CHANGE HISTORY**

20592 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20593 **Issue 6**20594 The [ENOSYS] error condition has been removed as stubs need not be provided if an
20595 implementation does not support the Asynchronous Input and Output option.

20596 The APPLICATION USAGE section is added.

20597 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
20598 CLOCK_MONOTONIC clock, if supported, is used.20599 **Issue 7**20600 The *aio_suspend()* function is moved from the Asynchronous Input and Output option to the
20601 Base.

20602 **NAME**

20603 aio_write — asynchronous write to a file

20604 **SYNOPSIS**

```
20605 #include <aio.h>
20606 int aio_write(struct aiocb *aiocbp);
```

20607 **DESCRIPTION**

20608 The *aio_write()* function shall write *aiocbp* ~~*aio_nbytes*~~ to the file associated with
 20609 *aiocbp* ~~*aio_fildes*~~ from the buffer pointed to by *aiocbp* ~~*aio_buf*~~. The function shall return when
 20610 the write request has been initiated or, at a minimum, queued to the file or device.

20611 PIO If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 20612 at a priority equal to a base scheduling priority minus *aiocbp* ~~*aio_reqprio*~~. If Thread Execution
 20613 Scheduling is not supported, then the base scheduling priority is that of the calling process;
 20614 PIO TPS otherwise, the base scheduling priority is that of the calling thread.

20615 The *aiocbp* argument may be used as an argument to *aio_error()* and *aio_return()* in order to
 20616 determine the error status and return status, respectively, of the asynchronous operation while it
 20617 is proceeding.

20618 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp* ~~*aio_buf*~~ or
 20619 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 20620 completion, then the behavior is undefined.

20621 If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take
 20622 place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called
 20623 immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to
 20624 SEEK_SET. If O_APPEND is set for the file descriptor, or if *aio_fildes* is associated with a device
 20625 that is incapable of seeking, write operations append to the file in the same order as the calls
 20626 were made, except under circumstances described in Section 2.8.2. After a successful call to
 20627 enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

20628 The *aio_sigevent* member specifies the notification which occurs when the request is completed.

20629 The *aiocbp* ~~*aio_lio_opcode*~~ field shall be ignored by *aio_write()*.

20630 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

20631 SIO If synchronized I/O is enabled on the file associated with *aiocbp* ~~*aio_fildes*~~, the behavior of this
 20632 function shall be according to the definitions of synchronized I/O data integrity completion, and
 20633 synchronized I/O file integrity completion.

20634 For any system action that changes the process memory space while an asynchronous I/O is
 20635 outstanding to the address range being changed, the result of that action is undefined.

20636 For regular files, no data transfer shall occur past the offset maximum established in the open
 20637 file description associated with *aiocbp* ~~*aio_fildes*~~.

20638 **RETURN VALUE**

20639 The *aio_write()* function shall return the value zero if the I/O operation is successfully queued;
 20640 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

20641 **ERRORS**

20642 The *aio_write()* function shall fail if:

20643 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 20644 resource limitations.

20645 Each of the following conditions may be detected synchronously at the time of the call to

20646 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the
 20647 *aio_write()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 20648 conditions below are detected asynchronously, the return status of the asynchronous operation
 20649 shall be set to `-1`, and the error status of the asynchronous operation is set to the corresponding
 20650 value.

20651 [EBADF] The *aio**bp* *aio_fildes* argument is not a valid file descriptor open for writing.

20652 [EINVAL] The file offset value implied by *aio**bp* *aio_offset* would be invalid,
 20653 PIO *aio**bp* *aio_reqprio* is not a valid value, or *aio**bp* *aio_nbytes* is an invalid
 20654 value.

20655 In the case that the *aio_write()* successfully queues the I/O operation, the return status of the
 20656 asynchronous operation shall be one of the values normally returned by the *write()* function call.
 20657 If the operation is successfully queued but is subsequently canceled or encounters an error, the
 20658 error status for the asynchronous operation contains one of the values normally set by the
 20659 *write()* function call, or one of the following:

20660 [EBADF] The *aio**bp* *aio_fildes* argument is not a valid file descriptor open for writing.

20661 [EINVAL] The file offset value implied by *aio**bp* *aio_offset* would be invalid.

20662 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 20663 *aio_cancel()* request.

20664 The following condition may be detected synchronously or asynchronously:

20665 [EFBIG] The file is a regular file, *aio**bp* *aio_nbytes* is greater than 0, and the starting
 20666 offset in *aio**bp* *aio_offset* is at or beyond the offset maximum in the open file
 20667 description associated with *aio**bp* *aio_fildes*.

20668 EXAMPLES

20669 None.

20670 APPLICATION USAGE

20671 None.

20672 RATIONALE

20673 None.

20674 FUTURE DIRECTIONS

20675 None.

20676 SEE ALSO

20677 Section 2.8.2 (on page 503), *aio_cancel()*, *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*,
 20678 *fork()*, *lio_listio()*, *lseek()*, *write()*

20679 XBD <[aio.h](#)>

20680 CHANGE HISTORY

20681 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

20682 Large File Summit extensions are added.

20683 Issue 6

20684 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 20685 implementation does not support the Asynchronous Input and Output option.

20686 The APPLICATION USAGE section is added.

20687 The following new requirements on POSIX implementations derive from alignment with the

- 20688 Single UNIX Specification:
- 20689 In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
20690 past the offset maximum established in the open file description associated with
20691 *aiocbp* ~~*aio_fildes*~~.
- 20692 The [EFBIG] error is added as part of the large file support extensions.
- 20693 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the
20694 DESCRIPTION when prioritized I/O is supported to account for threads, and removing the
20695 words “to the calling process” in the RETURN VALUE section.
- 20696 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL]
20697 error, so that detection of an [EINVAL] error for an invalid value of *aiocbp* ~~*aio_reqprio*~~ is only
20698 required if the Prioritized Input and Output option is supported.
- 20699 **Issue 7**
- 20700 Austin Group Interpretation 1003.1-2001 #082 is applied.
- 20701 The *aio_write()* function is moved from the Asynchronous Input and Output option to the Base.
20702 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0028 [317] is applied.

20703 **NAME**

20704 alarm ‡schedule an alarm signal

20705 **SYNOPSIS**

```
20706 #include <unistd.h>
20707 unsigned alarm(unsigned seconds);
```

20708 **DESCRIPTION**

20709 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after
 20710 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays
 20711 may prevent the process from handling the signal as soon as it is generated.

20712 If *seconds* is 0, a pending alarm request, if any, is canceled.

20713 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.
 20714 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time
 20715 at which the SIGALRM signal is generated.

20716 XSI Interactions between *alarm()* and *setitimer()* are unspecified.

20717 **RETURN VALUE**

20718 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value
 20719 that is the number of seconds until the previous request would have generated a SIGALRM
 20720 signal. Otherwise, *alarm()* shall return 0.

20721 **ERRORS**

20722 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

20723 **EXAMPLES**

20724 None.

20725 **APPLICATION USAGE**

20726 The *fork()* function clears pending alarms in the child process. A new process image created by
 20727 one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

20728 Application developers should note that the type of the argument *seconds* and the return value of
 20729 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces
 20730 Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX},
 20731 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting
 20732 its portability. A different type was considered, but historical implementations, including those
 20733 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

20734 Application developers should be aware of possible interactions when the same process uses
 20735 both the *alarm()* and *sleep()* functions.

20736 **RATIONALE**

20737 Many historical implementations (including Version 7 and System V) allow an alarm to occur up
 20738 to a second early. Other implementations allow alarms up to half a second or one clock tick
 20739 early or do not allow them to occur early at all. The latter is considered most appropriate, since it
 20740 gives the most predictable behavior, especially since the signal can always be delayed for an
 20741 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument
 20742 as the minimum amount of time they wish to have elapse before the signal.

20743 The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time
 20744 as common English usage, and has nothing to do with “realtime operating systems”. It is in
 20745 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

20746 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are
 20747 silently rounded down to an implementation-specific maximum value. This maximum is large

20748 enough (to the order of several months) that the effect is not noticeable.

20749 There were two possible choices for alarm generation in multi-threaded applications: generation
20750 for the calling thread or generation for the process. The first option would not have been
20751 particularly useful since the alarm state is maintained on a per-process basis and the alarm that
20752 is established by the last invocation of *alarm()* is the only one that would be active.

20753 Furthermore, allowing generation of an asynchronous signal for a thread would have
20754 introduced an exception to the overall signal model. This requires a compelling reason in order
20755 to be justified.

20756 **FUTURE DIRECTIONS**

20757 None.

20758 **SEE ALSO**

20759 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep()*

20760 XBD [<signal.h>](#), [<unistd.h>](#)

20761 **CHANGE HISTORY**

20762 First released in Issue 1. Derived from Issue 1 of the SVID.

20763 **Issue 6**

20764 The following new requirements on POSIX implementations derive from alignment with the
20765 Single UNIX Specification:

20766 The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*,
20767 and *usleep()* functions are unspecified.

20768 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/16 is applied, replacing “an
20769 implementation-defined maximum value” with “an implementation-specific maximum value”
20770 in the RATIONALE.

20771 **NAME**

20772 alphasort, scandir — scan a directory

20773 **SYNOPSIS**

```
20774       #include <dirent.h>
20775       int alphasort(const struct dirent **d1, const struct dirent **d2);
20776       int scandir(const char *dir, struct dirent ***namelist,
20777                   int (*sel)(const struct dirent *),
20778                   int (*compar)(const struct dirent **, const struct dirent **));
```

20779 **DESCRIPTION**

20780 The *alphasort()* function can be used as the comparison function for the *scandir()* function to sort
 20781 the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the
 20782 *strcoll()* function on the *d_name* element of the **dirent** structures passed as the two parameters. If
 20783 the *strcoll()* function fails, the return value of *alphasort()* is unspecified.

20784 The *alphasort()* function shall not change the setting of *errno* if successful. Since no return value
 20785 is reserved to indicate an error, an application wishing to check for error situations should set
 20786 *errno* to 0, then call *alphasort()*, then check *errno*.

20787 The *scandir()* function shall scan the directory *dir*, calling the function referenced by *sel* on each
 20788 directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored
 20789 in strings allocated as if by a call to *malloc()*, and sorted as if by a call to *qsort()* with the
 20790 comparison function *compar*, except that *compar* need not provide total ordering. The strings are
 20791 collected in array *namelist* which shall be allocated as if by a call to *malloc()*. If *sel* is a null
 20792 pointer, all entries shall be selected. If the comparison function *compar* does not provide total
 20793 ordering, the order in which the directory entries are stored is unspecified.

20794 **RETURN VALUE**

20795 Upon successful completion, the *alphasort()* function shall return an integer greater than, equal
 20796 to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is
 20797 lexically greater than, equal to, or less than the directory pointed to by *d2* when both are
 20798 interpreted as appropriate to the current locale. There is no return value reserved to indicate an
 20799 error.

20800 Upon successful completion, the *scandir()* function shall return the number of entries in the
 20801 array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir()*
 20802 function shall return -1.

20803 **ERRORS**20804 The *scandir()* function shall fail if:

20805 [EACCES] Search permission is denied for the component of the path prefix of *dir* or read
 20806 permission is denied for *dir*.

20807 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dir*
 20808 argument.

20809 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

20811 [ENOENT] A component of *dir* does not name an existing directory or *dir* is an empty
 20812 string.

20813 [ENOMEM] Insufficient storage space is available.

- 20814 [ENOTDIR] A component of *dir* names an existing file that is neither a directory nor a
20815 symbolic link to a directory.
- 20816 [EOVERFLOW] One of the values to be returned or passed to a callback function cannot be
20817 represented correctly.
- 20818 The *scandir()* function may fail if:
- 20819 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
20820 resolution of the *dir* argument.
- 20821 [EMFILE] All file descriptors available to the process are currently open.
- 20822 [ENAMETOOLONG]
20823 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
20824 symbolic link produced an intermediate result with a length that exceeds
20825 {PATH_MAX}.
- 20826 [ENFILE] Too many files are currently open in the system.

EXAMPLES

20827 An example to print the files in the current directory:

```
20829 #include <dirent.h>
20830 #include <stdio.h>
20831 #include <stdlib.h>
20832 ...
20833 struct dirent **namelist;
20834 int i,n;

20835     n = scandir(".", &namelist, 0, alphasort);
20836     if (n < 0)
20837         perror("scandir");
20838     else {
20839         for (i = 0; i < n; i++) {
20840             printf("%s\n", namelist[i]->d_name);
20841             free(namelist[i]);
20842         }
20843     }
20844     free(namelist);
20845     ...
```

APPLICATION USAGE

20846 If *dir* contains filenames that do not form character strings, or which contain characters outside
20847 the domain of the collating sequence of the current locale, the *alphasort()* function need not
20848 provide a total ordering. This condition is not possible if all filenames within the directory
20849 consist only of characters from the portable filename character set.

20851 The *scandir()* function may allocate dynamic storage during its operation. If *scandir()* is forcibly
20852 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *sel*
20853 or *compar*, or by an interrupt routine, *scandir()* does not have a chance to free that storage, so it
20854 remains permanently allocated. A safe way to handle interrupts is to store the fact that an
20855 interrupt has occurred, then wait until *scandir()* returns to act on the interrupt.

20856 For functions that allocate memory as if by *malloc()*, the application should release such memory
20857 when it is no longer required by a call to *free()*. For *scandir()*, this is *namelist* (including all of the
20858 individual strings in *namelist*).

20859 **RATIONALE**

20860 None.

20861 **FUTURE DIRECTIONS**

20862 None.

20863 **SEE ALSO**20864 *qsort()*, *strcoll()*20865 XBD <**dirent.h**>20866 **CHANGE HISTORY**

20867 First released in Issue 7.

20868 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0029 [324], XSH/TC1-2008/0030 [404],
20869 XSH/TC1-2008/0031 [393], and XSH/TC1-2008/0032 [291] are applied.

20870 **NAME**20871 asctime, asctime_r \dagger convert date and time to a string20872 **SYNOPSIS**

```
20873 OB #include <time.h>
20874 char *asctime(const struct tm *timeptr);
20875 OB CX char *asctime_r(const struct tm *restrict tm, char *restrict buf);
```

20876 **DESCRIPTION**

20877 CX For *asctime()*: The functionality described on this reference page is aligned with the ISO C
 20878 standard. Any conflict between the requirements described here and the ISO C standard is
 20879 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

20880 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*
 20881 into a string in the form:

```
20882 Sun Sep 16 01:03:52 1973\n\0
```

20883 using the equivalent of the following algorithm:

```
20884 char *asctime(const struct tm *timeptr)
20885 {
20886     static char wday_name[7][3] = {
20887         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
20888     };
20889     static char mon_name[12][3] = {
20890         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
20891         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
20892     };
20893     static char result[26];
20894     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
20895         wday_name[timeptr->tm_wday],
20896         mon_name[timeptr->tm_mon],
20897         timeptr->tm_mday, timeptr->tm_hour,
20898         timeptr->tm_min, timeptr->tm_sec,
20899         1900 + timeptr->tm_year);
20900     return result;
20901 }
```

20902 However, the behavior is undefined if *timeptr* \dagger *tm_wday* or *timeptr* \dagger *tm_mon* are not within the
 20903 normal ranges as defined in **<time.h>**, or if *timeptr* \dagger *tm_year* exceeds {INT_MAX}-1990, or if the
 20904 above algorithm would attempt to generate more than 26 bytes of output (including the
 20905 terminating null).

20906 The **tm** structure is defined in the **<time.h>** header.

20907 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 20908 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 20909 functions may overwrite the information returned in either of these objects by any of the other
 20910 functions.

20911 The *asctime()* function need not be thread-safe.

20912 The *asctime_r()* function shall convert the broken-down time in the structure pointed to by *tm*
 20913 into a string (of the same form as that returned by *asctime()*, and with the same undefined
 20914 behavior when input or output is out of range) that is placed in the user-supplied buffer pointed

20915 to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

20916 **RETURN VALUE**

20917 CX Upon successful completion, *asctime()* shall return a pointer to the string. If the function is
20918 unsuccessful, it shall return NULL.

20919 Upon successful completion, *asctime_r()* shall return a pointer to a character string containing
20920 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,
20921 it shall return NULL.

20922 **ERRORS**

20923 No errors are defined.

20924 **EXAMPLES**

20925 None.

20926 **APPLICATION USAGE**

20927 These functions are included only for compatibility with older implementations. They have
20928 undefined behavior if the resulting string would be too long, so the use of these functions
20929 should be discouraged. On implementations that do not detect output string length overflow, it
20930 is possible to overflow the output buffers in such a way as to cause applications to fail, or
20931 possible system security violations. Also, these functions do not support localized date and time
20932 formats. To avoid these problems, applications should use *strftime()* to generate strings from
20933 broken-down times.

20934 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

20935 The *asctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead
20936 of possibly using a static data area that may be overwritten by each call.

20937 **RATIONALE**

20938 The standard developers decided to mark the *asctime()* and *asctime_r()* functions obsolescent
20939 even though *asctime()* is in the ISO C standard due to the possibility of buffer overflow. The
20940 ISO C standard also provides the *strftime()* function which can be used to avoid these problems.

20941 **FUTURE DIRECTIONS**

20942 These functions may be removed in a future version.

20943 **SEE ALSO**

20944 [clock\(\)](#), [ctime\(\)](#), [difftime\(\)](#), [gmtime\(\)](#), [localtime\(\)](#), [mktime\(\)](#), [strftime\(\)](#), [strptime\(\)](#), [time\(\)](#), [utime\(\)](#)

20945 XBD [<time.h>](#)

20946 **CHANGE HISTORY**

20947 First released in Issue 1. Derived from Issue 1 of the SVID.

20948 **Issue 5**

20949 Normative text previously in the APPLICATION USAGE section is moved to the
20950 DESCRIPTION.

20951 The *asctime_r()* function is included for alignment with the POSIX Threads Extension.

20952 A note indicating that the *asctime()* function need not be reentrant is added to the
20953 DESCRIPTION.

20954 **Issue 6**

20955 The *asctime_r()* function is marked as part of the Thread-Safe Functions option.

20956 Extensions beyond the ISO C standard are marked.

20957 The APPLICATION USAGE section is updated to include a note on the thread-safe function and

- 20958 its avoidance of possibly using a static data area.
- 20959 The DESCRIPTION of *asctime_r()* is updated to describe the format of the string returned.
- 20960 The **restrict** keyword is added to the *asctime_r()* prototype for alignment with the
20961 ISO/IEC 9899:1999 standard
- 20962 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/17 is applied, adding the CX extension in
20963 the RETURN VALUE section requiring that if the *asctime()* function is unsuccessful it returns
20964 NULL.
- 20965 **Issue 7**
- 20966 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.
- 20967 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 20968 The *asctime_r()* function is moved from the Thread-Safe Functions option to the Base.
- 20969 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0033 [86,429] is applied.

20970 **NAME**20971 asin, asinf, asinl **arc sine function**20972 **SYNOPSIS**

```
20973 #include <math.h>
20974 double asin(double x);
20975 float asinf(float x);
20976 long double asinl(long double x);
```

20977 **DESCRIPTION**

20978 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 20979 conflict between the requirements described here and the ISO C standard is unintentional. This
 20980 volume of POSIX.1-2017 defers to the ISO C standard.

20981 These functions shall compute the principal value of the arc sine of their argument x . The value
 20982 of x should be in the range $[-1,1]$.

20983 An application wishing to check for error situations should set *errno* to zero and call
 20984 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20985 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20986 zero, an error has occurred.

20987 **RETURN VALUE**

20988 Upon successful completion, these functions shall return the arc sine of x , in the range
 20989 $[-\pi/2, \pi/2]$ radians.

20990 **MX** For finite values of x not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 20991 supported), or an implementation-defined value shall be returned.

20992 **MX** If x is NaN, a NaN shall be returned.

20993 If x is ± 0 , x shall be returned.

20994 If x is $\pm\text{Inf}$, a domain error shall occur, and a NaN shall be returned.

20995 If x is subnormal, a range error may occur

20996 **MXX** and x should be returned.

20997 **MX** If x is not returned, *asin()*, *asinf()*, and *asinl()* shall return an implementation-defined value no
 20998 greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

20999 **ERRORS**

21000 These functions shall fail if:

21001 **MX** **Domain Error** The x argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.

21002 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21003 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 21004 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 21005 shall be raised.

21006 These functions may fail if:

21007 **MX** **Range Error** The value of x is subnormal.

21008 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21009 then *errno* shall be set to [ERANGE]. If the integer expression
 21010 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 21011 floating-point exception shall be raised.

21012 **EXAMPLES**

21013 None.

21014 **APPLICATION USAGE**

21015 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
21016 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21017 **RATIONALE**

21018 None.

21019 **FUTURE DIRECTIONS**

21020 None.

21021 **SEE ALSO**21022 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [sin\(\)](#)21023 XBD [Section 4.20](#) (on page 117), [<math.h>](#)21024 **CHANGE HISTORY**

21025 First released in Issue 1. Derived from Issue 1 of the SVID.

21026 **Issue 5**

21027 The DESCRIPTION is updated to indicate how an application should check for an error. This
21028 text was previously published in the APPLICATION USAGE section.

21029 **Issue 6**21030 The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

21031 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21032 revised to align with the ISO/IEC 9899:1999 standard.

21033 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
21034 marked.

21035 **Issue 7**

21036 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0034 [320] and XSH/TC1-2008/0035
21037 [68] are applied.

21038 **NAME**

21039 asinh, asinhf, asinhl ‡inverse hyperbolic sine functions

21040 **SYNOPSIS**

```
21041 #include <math.h>
21042 double asinh(double x);
21043 float asinhf(float x);
21044 long double asinhl(long double x);
```

21045 **DESCRIPTION**

21046 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21047 conflict between the requirements described here and the ISO C standard is unintentional. This
 21048 volume of POSIX.1-2017 defers to the ISO C standard.

21049 These functions shall compute the inverse hyperbolic sine of their argument x .

21050 An application wishing to check for error situations should set *errno* to zero and call
 21051 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 21052 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 21053 zero, an error has occurred.

21054 **RETURN VALUE**

21055 Upon successful completion, these functions shall return the inverse hyperbolic sine of their
 21056 argument.

21057 MX If x is NaN, a NaN shall be returned.

21058 If x is ± 0 , or $\pm \text{Inf}$, x shall be returned.

21059 If x is subnormal, a range error may occur

21060 MXX and x should be returned.

21061 MX If x is not returned, *asinh()*, *asinhf()*, and *asinhl()* shall return an implementation-defined value
 21062 no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

21063 **ERRORS**

21064 These functions may fail if:

21065 MX **Range Error** The value of x is subnormal.

21066 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21067 then *errno* shall be set to [ERANGE]. If the integer expression
 21068 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 21069 floating-point exception shall be raised.

21070 **EXAMPLES**

21071 None.

21072 **APPLICATION USAGE**

21073 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 21074 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21075 **RATIONALE**

21076 None.

21077 **FUTURE DIRECTIONS**

21078 None.

21079 **SEE ALSO**21080 *feclearexcept()*, *fetestexcept()*, *sinh()*21081 XBD Section 4.20 (on page 117), `<math.h>`21082 **CHANGE HISTORY**

21083 First released in Issue 4, Version 2.

21084 **Issue 5**

21085 Moved from X/OPEN UNIX extension to BASE.

21086 **Issue 6**21087 The *asinh()* function is no longer marked as an extension.21088 The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999
21089 standard.21090 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21091 revised to align with the ISO/IEC 9899:1999 standard.21092 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
21093 marked.21094 **Issue 7**

21095 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0036 [68] is applied.

21096 **NAME**

21097 **asinl** long double sine function

21098 **SYNOPSIS**

21099 #include <math.h>

21100 long double asinl(long double x);

21101 **DESCRIPTION**

21102 Refer to *asin()*.

21103 **NAME**

21104 assert — insert program diagnostics

21105 **SYNOPSIS**

21106 #include <assert.h>

21107 void assert(*scalar expression*);21108 **DESCRIPTION**

21109 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 21110 conflict between the requirements described here and the ISO C standard is unintentional. This
 21111 volume of POSIX.1-2017 defers to the ISO C standard.

21112 The *assert()* macro shall insert diagnostics into programs; it shall expand to a **void** expression.
 21113 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal
 21114 to 0), *assert()* shall write information about the particular call that failed on *stderr* and shall call
 21115 *abort()*.

21116 The information written about the call that failed shall include the text of the argument, the
 21117 name of the source file, the source file line number, and the name of the enclosing function; the
 21118 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of
 21119 the identifier `__func__`.

21120 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the
 21121 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,
 21122 shall stop assertions from being compiled into the program.

21123 **RETURN VALUE**21124 The *assert()* macro shall not return a value.21125 **ERRORS**

21126 No errors are defined.

21127 **EXAMPLES**

21128 None.

21129 **APPLICATION USAGE**

21130 None.

21131 **RATIONALE**

21132 None.

21133 **FUTURE DIRECTIONS**

21134 None.

21135 **SEE ALSO**21136 *abort()*, *stdin*

21137 XBD <assert.h>

21138 **CHANGE HISTORY**

21139 First released in Issue 1. Derived from Issue 1 of the SVID.

21140 **Issue 6**

21141 The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment
 21142 with the ISO/IEC 9899:1999 standard.

21143 The DESCRIPTION of *assert()* is updated for alignment with the ISO/IEC 9899:1999 standard.

21144 **NAME**21145 atan, atanf, atanl \ddagger arc tangent function21146 **SYNOPSIS**

```
21147 #include <math.h>
21148 double atan(double x);
21149 float atanf(float x);
21150 long double atanl(long double x);
```

21151 **DESCRIPTION**

21152 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21153 conflict between the requirements described here and the ISO C standard is unintentional. This
 21154 volume of POSIX.1-2017 defers to the ISO C standard.

21155 These functions shall compute the principal value of the arc tangent of their argument x .

21156 An application wishing to check for error situations should set *errno* to zero and call
 21157 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 21158 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 21159 zero, an error has occurred.

21160 **RETURN VALUE**

21161 Upon successful completion, these functions shall return the arc tangent of x in the range
 21162 $[-\pi/2, \pi/2]$ radians.

21163 MX If x is NaN, a NaN shall be returned.

21164 If x is ± 0 , x shall be returned.

21165 If x is $\pm \text{Inf}$, $\pm \pi/2$ shall be returned.

21166 If x is subnormal, a range error may occur

21167 MXX and x should be returned.

21168 MX If x is not returned, *atan()*, *atanf()*, and *atanl()* shall return an implementation-defined value no
 21169 greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

21170 **ERRORS**

21171 These functions may fail if:

21172 MX **Range Error** The value of x is subnormal.

21173 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21174 then *errno* shall be set to [ERANGE]. If the integer expression
 21175 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 21176 floating-point exception shall be raised.

21177 **EXAMPLES**

21178 None.

21179 **APPLICATION USAGE**

21180 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 21181 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21182 **RATIONALE**

21183 None.

21184 **FUTURE DIRECTIONS**

21185 None.

21186 **SEE ALSO**21187 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*21188 XBD Section 4.20 (on page 117), **<math.h>**21189 **CHANGE HISTORY**

21190 First released in Issue 1. Derived from Issue 1 of the SVID.

21191 **Issue 5**21192 The DESCRIPTION is updated to indicate how an application should check for an error. This
21193 text was previously published in the APPLICATION USAGE section.21194 **Issue 6**21195 The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.21196 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21197 revised to align with the ISO/IEC 9899:1999 standard.21198 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
21199 marked.21200 **Issue 7**

21201 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0037 [68] is applied.

21202 **NAME**21203 atan2, atan2f, atan2l \dagger arc tangent functions21204 **SYNOPSIS**

```
21205 #include <math.h>
21206 double atan2(double y, double x);
21207 float atan2f(float y, float x);
21208 long double atan2l(long double y, long double x);
```

21209 **DESCRIPTION**

21210 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21211 conflict between the requirements described here and the ISO C standard is unintentional. This
 21212 volume of POSIX.1-2017 defers to the ISO C standard.

21213 These functions shall compute the principal value of the arc tangent of y/x , using the signs of
 21214 both arguments to determine the quadrant of the return value.

21215 An application wishing to check for error situations should set *errno* to zero and call
 21216 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 21217 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 21218 zero, an error has occurred.

21219 **RETURN VALUE**

21220 Upon successful completion, these functions shall return the arc tangent of y/x in the range
 21221 $[-\pi, \pi]$ radians.

21222 If y is ± 0 and x is < 0 , $\pm\pi$ shall be returned.

21223 If y is ± 0 and x is > 0 , ± 0 shall be returned.

21224 If y is < 0 and x is ± 0 , $-\pi/2$ shall be returned.

21225 If y is > 0 and x is ± 0 , $\pi/2$ shall be returned.

21226 If x is 0, a pole error shall not occur.

21227 MX If either x or y is NaN, a NaN shall be returned.

21228 If the correct value would cause underflow, a range error may occur, and *atan()*, *atan2f()*, and
 21229 *atan2l()* shall return an implementation-defined value no greater in magnitude than DBL_MIN,
 21230 FLT_MIN, and LDBL_MIN, respectively.

21231 MXX If the IEC 60559 Floating-Point option is supported, y/x should be returned.

21232 MX If y is ± 0 and x is -0 , $\pm\pi$ shall be returned.

21233 If y is ± 0 and x is $+0$, ± 0 shall be returned.

21234 For finite values of $\pm y > 0$, if x is $-\text{Inf}$, $\pm\pi$ shall be returned.

21235 For finite values of $\pm y > 0$, if x is $+\text{Inf}$, ± 0 shall be returned.

21236 For finite values of x , if y is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.

21237 If y is $\pm\text{Inf}$ and x is $-\text{Inf}$, $\pm 3\pi/4$ shall be returned.

21238 If y is $\pm\text{Inf}$ and x is $+\text{Inf}$, $\pm\pi/4$ shall be returned.

21239 If both arguments are 0, a domain error shall not occur.

21240 **ERRORS**

21241 These functions may fail if:

21242 MX **Range Error** The result underflows.

21243 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21244 then *errno* shall be set to [ERANGE]. If the integer expression
 21245 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 21246 floating-point exception shall be raised.

21247 **EXAMPLES**21248 **Converting Cartesian to Polar Coordinates System**

21249 The function below uses *atan2()* to convert a 2d vector expressed in cartesian coordinates (*x,y*) to
 21250 the polar coordinates (*rho,theta*). There are other ways to compute the angle *theta*, using *asin()*
 21251 *acos()*, or *atan()*. However, *atan2()* presents here two advantages:

21252 The angle's quadrant is automatically determined.

21253 The singular cases (0,*y*) are taken into account.

21254 Finally, this example uses *hypot()* rather than *sqrt()* since it is better for special cases; see *hypot()*
 21255 for more information.

```
21256 #include <math.h>
21257 void
21258 cartesian_to_polar(const double x, const double y,
21259                  double *rho, double *theta
21260                  )
21261 {
21262     *rho    = hypot (x,y); /* better than sqrt(x*x+y*y) */
21263     *theta  = atan2 (y,x);
21264 }
```

21265 **APPLICATION USAGE**

21266 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 21267 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21268 **RATIONALE**

21269 None.

21270 **FUTURE DIRECTIONS**

21271 None.

21272 **SEE ALSO**21273 [acos\(\)](#), [asin\(\)](#), [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [hypot\(\)](#), [isnan\(\)](#), [sqrt\(\)](#), [tan\(\)](#)21274 XBD [Section 4.20](#) (on page 117), [<math.h>](#)21275 **CHANGE HISTORY**

21276 First released in Issue 1. Derived from Issue 1 of the SVID.

21277 **Issue 5**

21278 The DESCRIPTION is updated to indicate how an application should check for an error. This
 21279 text was previously published in the APPLICATION USAGE section.

21280 **Issue 6**

21281 The *atan2f()* and *atan2l()* functions are added for alignment with the ISO/IEC 9899:1999
21282 standard.

21283 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
21284 revised to align with the ISO/IEC 9899:1999 standard, and the IEC 60559:1989 standard
21285 floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

21286 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/18 is applied, adding to the EXAMPLES
21287 section.

21288 **Issue 7**

21289 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0038 [68,428] is applied.

21290 **NAME**

21291 atanf arctangent function

21292 **SYNOPSIS**

21293 #include <math.h>

21294 float atanf(float x);

21295 **DESCRIPTION**

21296 Refer to *atan()*.

21297 **NAME**

21298 atanh, atanhf, atanh1 ‡inverse hyperbolic tangent functions

21299 **SYNOPSIS**

```
21300       #include <math.h>
21301       double atanh(double x);
21302       float atanhf(float x);
21303       long double atanh1(long double x);
```

21304 **DESCRIPTION**

21305 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21306 conflict between the requirements described here and the ISO C standard is unintentional. This
 21307 volume of POSIX.1-2017 defers to the ISO C standard.

21308 These functions shall compute the inverse hyperbolic tangent of their argument x .

21309 An application wishing to check for error situations should set *errno* to zero and call
 21310 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 21311 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 21312 zero, an error has occurred.

21313 **RETURN VALUE**

21314 Upon successful completion, these functions shall return the inverse hyperbolic tangent of their
 21315 argument.

21316 If x is ± 1 , a pole error shall occur, and *atanh()*, *atanhf()*, and *atanh1()* shall return the value of the
 21317 macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the
 21318 correct value of the function.

21319 MX For finite $|x| > 1$, a domain error shall occur, and either a NaN (if supported), or an
 21320 implementation-defined value shall be returned.

21321 MX If x is NaN, a NaN shall be returned.

21322 If x is ± 0 , x shall be returned.

21323 If x is $\pm \text{Inf}$, a domain error shall occur, and a NaN shall be returned.

21324 If x is subnormal, a range error may occur

21325 MXX and x should be returned.

21326 MX If x is not returned, *atanh()*, *atanhf()*, and *atanh1()* shall return an implementation-defined value
 21327 no greater in magnitude than DBL_MIN, FLT_MIN, and LDL_MIN, respectively.

21328 **ERRORS**

21329 These functions shall fail if:

21330 MX Domain Error The x argument is finite and not in the range $[-1, 1]$, or is $\pm \text{Inf}$.

21331 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21332 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 21333 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 21334 shall be raised.

21335 Pole Error The x argument is ± 1 .

21336 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21337 then *errno* shall be set to [ERANGE]. If the integer expression
 21338 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 21339 floating-point exception shall be raised.

21340 These functions may fail if:

21341 MX **Range Error** The value of x is subnormal.

21342 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 21343 then *errno* shall be set to [ERANGE]. If the integer expression
 21344 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 21345 floating-point exception shall be raised.

21346 **EXAMPLES**

21347 None.

21348 **APPLICATION USAGE**

21349 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 21350 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21351 **RATIONALE**

21352 None.

21353 **FUTURE DIRECTIONS**

21354 None.

21355 **SEE ALSO**

21356 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [tanh\(\)](#)

21357 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

21358 **CHANGE HISTORY**

21359 First released in Issue 4, Version 2.

21360 **Issue 5**

21361 Moved from X/OPEN UNIX extension to BASE.

21362 **Issue 6**

21363 The *atanh()* function is no longer marked as an extension.

21364 The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999
 21365 standard.

21366 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 21367 revised to align with the ISO/IEC 9899:1999 standard.

21368 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 21369 marked.

21370 **Issue 7**

21371 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0039 [320] and XSH/TC1-2008/0040
 21372 [680] are applied.

21373 **NAME**

21374 atanl arctangent function

21375 **SYNOPSIS**

21376 #include <math.h>

21377 long double atanl(long double x);

21378 **DESCRIPTION**

21379 Refer to *atan()*.

21380 **NAME**

21381 atexit — register a function to run at process termination

21382 **SYNOPSIS**

21383 #include <stdlib.h>

21384 int atexit(void (*func)(void));

21385 **DESCRIPTION**

21386 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21387 conflict between the requirements described here and the ISO C standard is unintentional. This
 21388 volume of POSIX.1-2017 defers to the ISO C standard.

21389 The *atexit()* function shall register the function pointed to by *func*, to be called without
 21390 arguments at normal program termination. At normal program termination, all functions
 21391 registered by the *atexit()* function shall be called, in the reverse order of their registration, except
 21392 that a function is called after any previously registered functions that had already been called at
 21393 the time it was registered. Normal termination occurs either by a call to *exit()* or a return from
 21394 *main()*.

21395 At least 32 functions can be registered with *atexit()*.

21396 CX After a successful call to any of the *exec* functions, any functions previously registered by *atexit()*
 21397 shall no longer be registered.

21398 **RETURN VALUE**21399 Upon successful completion, *atexit()* shall return 0; otherwise, it shall return a non-zero value.21400 **ERRORS**

21401 No errors are defined.

21402 **EXAMPLES**

21403 None.

21404 **APPLICATION USAGE**21405 The functions registered by a call to *atexit()* must return to ensure that all registered functions
 21406 are called.

21407 The application should call *sysconf()* to obtain the value of {ATEXIT_MAX}, the number of
 21408 functions that can be registered. There is no way for an application to tell how many functions
 21409 have already been registered with *atexit()*.

21410 Since the behavior is undefined if the *exit()* function is called more than once, portable
 21411 applications calling *atexit()* must ensure that the *exit()* function is not called at normal process
 21412 termination when all functions registered by the *atexit()* function are called.

21413 All functions registered by the *atexit()* function are called at normal process termination, which
 21414 occurs by a call to the *exit()* function or a return from *main()* or on the last thread termination,
 21415 when the behavior is as if the implementation called *exit()* with a zero argument at thread
 21416 termination time.

21417 If, at normal process termination, a function registered by the *atexit()* function is called and a
 21418 portable application needs to stop further *exit()* processing, it must call the *_exit()* function or
 21419 the *_Exit()* function or one of the functions which cause abnormal process termination.

21420 **RATIONALE**

21421 None.

21422 **FUTURE DIRECTIONS**

21423 None.

21424 **SEE ALSO**21425 *exec*, *exit()*, *sysconf()*

21426 XBD <stdlib.h>

21427 **CHANGE HISTORY**

21428 First released in Issue 4. Derived from the ANSI C standard.

21429 **Issue 6**

21430 Extensions beyond the ISO C standard are marked.

21431 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

21432 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/19 is applied, adding further clarification
21433 to the APPLICATION USAGE section.

21434 **NAME**

21435 atof — convert a string to a double-precision number

21436 **SYNOPSIS**

21437 #include <stdlib.h>

21438 double atof(const char *str);

21439 **DESCRIPTION**

21440 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21441 conflict between the requirements described here and the ISO C standard is unintentional. This
21442 volume of POSIX.1-2017 defers to the ISO C standard.

21443 The call *atof(str)* shall be equivalent to:

21444 strtod(str, (char **)NULL),

21445 except that the handling of errors may differ. If the value cannot be represented, the behavior is
21446 undefined.21447 **RETURN VALUE**21448 The *atof()* function shall return the converted value if the value can be represented.21449 **ERRORS**

21450 No errors are defined.

21451 **EXAMPLES**

21452 None.

21453 **APPLICATION USAGE**

21454 The *atof()* function is subsumed by *strtod()* but is retained because it is used extensively in
21455 existing code. If the number is not known to be in range, *strtod()* should be used because *atof()*
21456 is not required to perform any error checking.

21457 **RATIONALE**

21458 None.

21459 **FUTURE DIRECTIONS**

21460 None.

21461 **SEE ALSO**21462 [strtod\(\)](#)21463 XBD [<stdlib.h>](#)21464 **CHANGE HISTORY**

21465 First released in Issue 1. Derived from Issue 1 of the SVID.

21466 **NAME**

21467 atoi ¶convert a string to an integer

21468 **SYNOPSIS**

21469 #include <stdlib.h>

21470 int atoi(const char *str);

21471 **DESCRIPTION**

21472 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21473 conflict between the requirements described here and the ISO C standard is unintentional. This
 21474 volume of POSIX.1-2017 defers to the ISO C standard.

21475 The call *atoi(str)* shall be equivalent to:

21476 (int) strtol(str, (char **)NULL, 10)

21477 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 21478 undefined.

21479 **RETURN VALUE**21480 The *atoi()* function shall return the converted value if the value can be represented.21481 **ERRORS**

21482 No errors are defined.

21483 **EXAMPLES**21484 **Converting an Argument**

21485 The following example checks for proper usage of the program. If there is an argument and the
 21486 decimal conversion of this argument (obtained using *atoi()*) is greater than 0, then the program
 21487 has a valid number of minutes to wait for an event.

21488 #include <stdlib.h>

21489 #include <stdio.h>

21490 ...

21491 int minutes_to_event;

21492 ...

21493 if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {

21494 fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);

21495 }

21496 ...

21497 **APPLICATION USAGE**

21498 The *atoi()* function is subsumed by *strtol()* but is retained because it is used extensively in
 21499 existing code. If the number is not known to be in range, *strtol()* should be used because *atoi()* is
 21500 not required to perform any error checking.

21501 **RATIONALE**

21502 None.

21503 **FUTURE DIRECTIONS**

21504 None.

21505 **SEE ALSO**21506 [strtol\(\)](#)21507 XBD [<stdlib.h>](#)



21508
21509

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.



21510 **NAME**21511 `atol, atoll` ‡convert a string to a long integer21512 **SYNOPSIS**21513 `#include <stdlib.h>`21514 `long atol(const char *nptr);`21515 `long long atoll(const char *nptr);`21516 **DESCRIPTION**

21517 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21518 conflict between the requirements described here and the ISO C standard is unintentional. This
 21519 volume of POSIX.1-2017 defers to the ISO C standard.

21520 Except as noted below, the call `atol(nptr)` shall be equivalent to:21521 `strtol(nptr, (char **)NULL, 10)`21522 Except as noted below, the call to `atoll(nptr)` shall be equivalent to:21523 `strtoll(nptr, (char **)NULL, 10)`

21524 The handling of errors may differ. If the value cannot be represented, the behavior is undefined.

21525 **RETURN VALUE**

21526 These functions shall return the converted value if the value can be represented.

21527 **ERRORS**

21528 No errors are defined.

21529 **EXAMPLES**

21530 None.

21531 **APPLICATION USAGE**

21532 If the number is not known to be in range, `strtol()` or `strtoll()` should be used because `atol()` and
 21533 `atoll()` are not required to perform any error checking.

21534 **RATIONALE**

21535 None.

21536 **FUTURE DIRECTIONS**

21537 None.

21538 **SEE ALSO**21539 [strtol\(\)](#)21540 XBD [<stdlib.h>](#)21541 **CHANGE HISTORY**

21542 First released in Issue 1. Derived from Issue 1 of the SVID.

21543 **Issue 6**21544 The `atoll()` function is added for alignment with the ISO/IEC 9899:1999 standard.21545 **Issue 7**21546 SD5-XSH-ERN-61 is applied, correcting the DESCRIPTION of `atoll()`.

21547 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0046 [892] is applied.

21548 **NAME**

21549 basename — return the last component of a pathname

21550 **SYNOPSIS**

```
21551 XSI      #include <libgen.h>
21552         char *basename(char *path);
```

21553 **DESCRIPTION**

21554 The *basename()* function shall take the pathname pointed to by *path* and return a pointer to the
21555 final component of the pathname, deleting any trailing '/' characters.

21556 If the string pointed to by *path* consists entirely of the '/' character, *basename()* shall return a
21557 pointer to the string "/". If the string pointed to by *path* is exactly "/", it is implementation-
21558 defined whether '/' or "/" is returned.

21559 If *path* is a null pointer or points to an empty string, *basename()* shall return a pointer to the
21560 string ".".

21561 The *basename()* function may modify the string pointed to by *path*, and may return a pointer to
21562 internal storage. The returned pointer might be invalidated or the storage might be overwritten
21563 by a subsequent call to *basename()*. The returned pointer might also be invalidated if the calling
21564 thread is terminated.

21565 The *basename()* function need not be thread-safe.

21566 **RETURN VALUE**

21567 The *basename()* function shall return a pointer to the final component of *path*.

21568 **ERRORS**

21569 No errors are defined.

21570 **EXAMPLES**21571 **Using basename()**

21572 The following program fragment returns a pointer to the value *lib*, which is the base name of
21573 */usr/lib*.

```
21574         #include <libgen.h>
21575         ...
21576         char name[] = "/usr/lib";
21577         char *base;
21578
21578         base = basename(name);
21579         ...
```

21580 **Sample Input and Output Strings for the basename() and dirname() Functions and the**
21581 **basename and dirname Utilities**

	basename() and dirname() Functions path Argument	String Returned by basename()	String Returned by dirname()	basename and dirname Utilities string Operand	Output Written by basename Utility	Output Written by dirname Utility
21582	"usr"	"usr"	". "	usr	usr	.
21583	"usr/"	"usr"	". "	usr/	usr	.
21584	" "	". "	". "	"	. or empty string	.
21585	"/"	"/"	"/"	/	/	/
21586	"//"	"/" or "//"	"/" or "//"	//	/ or //	/ or //
21587	"///"	"/"	"/"	///	/	/
21588	"/usr/"	"usr"	"/"	/usr/	usr	/
21589	"/usr/lib"	"lib"	"/usr"	/usr/lib	lib	/usr
21590	"//usr//lib//"	"lib"	"//usr"	//usr//lib//	lib	//usr
21591	"/home//dwc//test"	"test"	"/home//dwc"	/home//dwc//test	test	/home//dwc
21592						

21597 APPLICATION USAGE

21598 None.

21599 RATIONALE

21600 None.

21601 FUTURE DIRECTIONS

21602 None.

21603 SEE ALSO

21604 [dirname\(\)](#)

21605 XBD [<libgen.h>](#)

21606 XCU [basename](#)

21607 CHANGE HISTORY

21608 First released in Issue 4, Version 2.

21609 Issue 5

21610 Moved from X/OPEN UNIX extension to BASE.

21611 Normative text previously in the APPLICATION USAGE section is moved to the
21612 DESCRIPTION.

21613 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

21614 Issue 6

21615 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

21616 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/20 is applied, changing the
21617 DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.

21618 Issue 7

21619 Austin Group Interpretation 1003.1-2001 #156 is applied.

21620 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0041 [75] is applied.

21621 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0047 [656], XSH/TC2-2008/0048 [928],
21622 and XSH/TC2-2008/0049 [612] are applied.

21623 **NAME**

21624 bind †bind a name to a socket

21625 **SYNOPSIS**

```
21626       #include <sys/socket.h>
21627       int bind(int socket, const struct sockaddr *address,
21628               socklen_t address_len);
```

21629 **DESCRIPTION**

21630 The *bind()* function shall assign a local socket address *address* to a socket identified by descriptor
 21631 *socket* that has no local socket address assigned. Sockets created with the *socket()* function are
 21632 initially unnamed; they are identified only by their address family.

21633 The *bind()* function takes the following arguments:

21634	<i>socket</i>	Specifies the file descriptor of the socket to be bound.
21635	<i>address</i>	Points to a sockaddr structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.
21636		
21637		
21638	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> argument.
21639		

21640 The socket specified by *socket* may require the process to have appropriate privileges to use the
 21641 *bind()* function.

21642 If the address family of the socket is AF_UNIX and the pathname in *address* names a symbolic
 21643 link, *bind()* shall fail and set *errno* to [EADDRINUSE].

21644 If the socket address cannot be assigned immediately and O_NONBLOCK is set for the file
 21645 descriptor for the socket, *bind()* shall fail and set *errno* to [EINPROGRESS], but the assignment
 21646 request shall not be aborted, and the assignment shall be completed asynchronously. Subsequent
 21647 calls to *bind()* for the same socket, before the assignment is completed, shall fail and set *errno* to
 21648 [EALREADY].

21649 When the assignment has been performed asynchronously, *pselect()*, *select()*, and *poll()* shall
 21650 indicate that the file descriptor for the socket is ready for reading and writing.

21651 **RETURN VALUE**

21652 Upon successful completion, *bind()* shall return 0; otherwise, -1 shall be returned and *errno* set
 21653 to indicate the error.

21654 **ERRORS**

21655 The *bind()* function shall fail if:

21656	[EADDRINUSE]	The specified address is already in use.
21657	[EADDRNOTAVAIL]	
21658		The specified address is not available from the local machine.
21659	[EAFNOSUPPORT]	
21660		The specified address is not a valid address for the address family of the specified socket.
21661		
21662	[EALREADY]	An assignment request is already in progress for the specified socket.
21663	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.

21664	[EINPROGRESS]	O_NONBLOCK is set for the file descriptor for the socket and the assignment cannot be immediately performed; the assignment shall be performed asynchronously.
21665		
21666		
21667	[EINVAL]	The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.
21668		
21669	[ENOBUFS]	Insufficient resources were available to complete the call.
21670	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
21671	[EOPNOTSUPP]	The socket type of the specified socket does not support binding to an address.
21672		If the address family of the socket is AF_UNIX, then <i>bind()</i> shall fail if:
21673	[EACCES]	A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission.
21674		
21675		
21676	[EDESTADDRREQ] or [EISDIR]	The <i>address</i> argument is a null pointer.
21677		
21678	[EIO]	An I/O error occurred.
21679	[ELOOP]	A loop exists in symbolic links encountered during resolution of the pathname in <i>address</i> .
21680		
21681	[ENAMETOOLONG]	The length of a component of a pathname is longer than {NAME_MAX}.
21682		
21683	[ENOENT]	A component of the path prefix of the pathname in <i>address</i> does not name an existing file or the pathname is an empty string.
21684		
21685	[ENOENT] or [ENOTDIR]	The pathname in <i>address</i> contains at least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i> characters. If the pathname without the trailing <i><slash></i> characters would name an existing file, an [ENOENT] error shall not occur.
21686		
21687		
21688		
21689		
21690	[ENOTDIR]	A component of the path prefix of the pathname in <i>address</i> names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in <i>address</i> contains at least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
21691		
21692		
21693		
21694		
21695		
21696	[EROFS]	The name would reside on a read-only file system.
21697		The <i>bind()</i> function may fail if:
21698	[EACCES]	The specified address is protected and the current user does not have permission to bind to it.
21699		
21700	[EINVAL]	The <i>address_len</i> argument is not a valid length for the address family.
21701	[EISCONN]	The socket is already connected.
21702	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in <i>address</i> .
21703		

21704 [ENAMETOOLONG]
 21705 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 21706 symbolic link produced an intermediate result with a length that exceeds
 21707 {PATH_MAX}.

EXAMPLES

21708 The following code segment shows how to create a socket and bind it to a name in the AF_UNIX
 21709 domain.

```
21710 #define MY_SOCKET_PATH "/somepath"
21711
21712 int sfd;
21713 struct sockaddr_un my_addr;
21714
21715 sfd = socket(AF_UNIX, SOCK_STREAM, 0);
21716 if (sfd == -1)
21717     /* Handle error */;
21718
21719 memset(&my_addr, '\0', sizeof(struct sockaddr_un));
21720 /* Clear structure */
21721 my_addr.sun_family = AF_UNIX;
21722 strncpy(my_addr.sun_path, MY_SOCKET_PATH, sizeof(my_addr.sun_path) - 1);
21723
21724 if (bind(sfd, (struct sockaddr *) &my_addr,
21725         sizeof(struct sockaddr_un)) == -1)
21726     /* Handle error */;
```

APPLICATION USAGE

21724 An application program can retrieve the assigned socket name with the *getsockname()* function.

RATIONALE

21726 None.

FUTURE DIRECTIONS

21728 None.

SEE ALSO

21730 *connect()*, *getsockname()*, *listen()*, *socket()*

21732 XBD <[sys/socket.h](#)>

CHANGE HISTORY

21733 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

21736 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS]
 21737 error to become a “shall fail” error.

21738 Austin Group Interpretation 1003.1-2001 #143 is applied.

21739 SD5-XSH-ERN-185 is applied.

21740 An example is added.

21741 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0042 [146], XSH/TC1-2008/0043 [146],
 21742 and XSH/TC1-2008/0044 [324] are applied.

21743 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0050 [822] is applied.

21744 **NAME**21745 `bsearch` — binary search a sorted table21746 **SYNOPSIS**21747 `#include <stdlib.h>`21748 `void *bsearch(const void *key, const void *base, size_t nel,`
21749 `size_t width, int (*compar)(const void *, const void *));`21750 **DESCRIPTION**21751 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21752 conflict between the requirements described here and the ISO C standard is unintentional. This
21753 volume of POSIX.1-2017 defers to the ISO C standard.21754 The `bsearch()` function shall search an array of `nel` objects, the initial element of which is pointed
21755 to by `base`, for an element that matches the object pointed to by `key`. The size of each element in
21756 the array is specified by `width`. If the `nel` argument has the value zero, the comparison function
21757 pointed to by `compar` shall not be called and no match shall be found.21758 The comparison function pointed to by `compar` shall be called with two arguments that point to
21759 the `key` object and to an array element, in that order.21760 The application shall ensure that the comparison function pointed to by `compar` does not alter the
21761 contents of the array. The implementation may reorder elements of the array between calls to the
21762 comparison function, but shall not alter the contents of any individual element.

21763 The implementation shall ensure that the first argument is always a pointer to the key.

21764 When the same objects (consisting of `width` bytes, irrespective of their current positions in the
21765 array) are passed more than once to the comparison function, the results shall be consistent with
21766 one another. That is, the same object shall always compare the same way with the key.21767 The application shall ensure that the function returns an integer less than, equal to, or greater
21768 than 0 if the `key` object is considered, respectively, to be less than, to match, or to be greater than
21769 the array element. The application shall ensure that the array consists of all the elements that
21770 compare less than, all the elements that compare equal to, and all the elements that compare
21771 greater than the `key` object, in that order.21772 **RETURN VALUE**21773 The `bsearch()` function shall return a pointer to a matching member of the array, or a null pointer
21774 if no match is found. If two or more members compare equal, which member is returned is
21775 unspecified.21776 **ERRORS**

21777 No errors are defined.

21778 **EXAMPLES**21779 The example below searches a table containing pointers to nodes consisting of a string and its
21780 length. The table is ordered alphabetically on the string in the node pointed to by each entry.21781 The code fragment below reads in strings and either finds the corresponding node and prints
21782 out the string and its length, or prints an error message.21783 `#include <stdio.h>`
21784 `#include <stdlib.h>`
21785 `#include <string.h>`

21786 `#define TABSIZE 1000`
21787 `struct node { /* These are stored in the table. */`

```

21788     char *string;
21789     int length;
21790 };
21791 struct node table[TABSIZE];    /* Table to be searched. */
21792     .
21793     .
21794     .
21795 {
21796     struct node *node_ptr, node;
21797     /* Routine to compare 2 nodes. */
21798     int node_compare(const void *, const void *);
21799     .
21800     .
21801     .
21802     while (scanf("%ms", &node.string) != EOF) {
21803         node_ptr = (struct node *)bsearch((void *)&node,
21804             (void *)table, TABSIZE,
21805             sizeof(struct node), node_compare);
21806         if (node_ptr != NULL) {
21807             (void)printf("string = %20s, length = %d\n",
21808                 node_ptr->string, node_ptr->length);
21809         } else {
21810             (void)printf("not found: %s\n", node.string);
21811         }
21812         free(node.string);
21813     }
21814 }
21815 /*
21816     This routine compares two nodes based on an
21817     alphabetical ordering of the string field.
21818 */
21819 int
21820 node_compare(const void *node1, const void *node2)
21821 {
21822     return strcoll(((const struct node *)node1)->string,
21823         ((const struct node *)node2)->string);
21824 }

```

21825 APPLICATION USAGE

21826 The pointers to the key and the element at the base of the table should be of type pointer-to-
21827 element.

21828 The comparison function need not compare every byte, so arbitrary data may be contained in
21829 the elements in addition to the values being compared.

21830 In practice, the array is usually sorted according to the comparison function.

21831 RATIONALE

21832 The requirement that the second argument (hereafter referred to as *p*) to the comparison function
21833 is a pointer to an element of the array implies that for every call all of the following expressions
21834 are non-zero:

```

21835 ( (char *)p - (char *)base ) % width == 0
21836 (char *)p >= (char *)base

```

21837 (char *)p < (char *)base + nel * width

21838 **FUTURE DIRECTIONS**

21839 None.

21840 **SEE ALSO**

21841 *hcreate(), lsearch(), qsort(), tdelete()*

21842 XBD <stdlib.h>

21843 **CHANGE HISTORY**

21844 First released in Issue 1. Derived from Issue 1 of the SVID.

21845 **Issue 6**

21846 The normative text is updated to avoid use of the term “must” for application requirements.

21847 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the
21848 DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three
21849 paragraphs. The RATIONALE section is also updated. These changes are for alignment with the
21850 ISO C standard.

21851 **Issue 7**

21852 The EXAMPLES section is revised.

21853 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0051 [756] is applied.

21854 **NAME**

21855 btowc ‡single byte to wide character conversion

21856 **SYNOPSIS**

21857 #include <stdio.h>

21858 #include <wchar.h>

21859 wint_t btowc(int c);

21860 **DESCRIPTION**

21861 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21862 conflict between the requirements described here and the ISO C standard is unintentional. This
 21863 volume of POSIX.1-2017 defers to the ISO C standard.

21864 The *btowc()* function shall determine whether *c* constitutes a valid (one-byte) character in the
 21865 initial shift state.

21866 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

21867 **RETURN VALUE**

21868 The *btowc()* function shall return WEOF if *c* has the value EOF or if (**unsigned char**) *c* does not
 21869 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the
 21870 wide-character representation of that character.

21871 CX In the POSIX locale, *btowc()* shall not return WEOF if *c* has a value in the range 0 to 255
 21872 inclusive.

21873 **ERRORS**

21874 No errors are defined.

21875 **EXAMPLES**

21876 None.

21877 **APPLICATION USAGE**

21878 None.

21879 **RATIONALE**

21880 None.

21881 **FUTURE DIRECTIONS**

21882 None.

21883 **SEE ALSO**21884 *wctob()*

21885 XBD <stdio.h>, <wchar.h>

21886 **CHANGE HISTORY**

21887 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 21888 (E).

21889 **Issue 7**

21890 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0052 [663] is applied.

21891 **NAME**

21892 cabs, cabsf, cabsl — return a complex absolute value

21893 **SYNOPSIS**

21894 #include <complex.h>

21895 double cabs(double complex z);

21896 float cabsf(float complex z);

21897 long double cabsl(long double complex z);

21898 **DESCRIPTION**

21899 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21900 conflict between the requirements described here and the ISO C standard is unintentional. This
21901 volume of POSIX.1-2017 defers to the ISO C standard.

21902 These functions shall compute the complex absolute value (also called norm, modulus, or
21903 magnitude) of z.

21904 **RETURN VALUE**

21905 These functions shall return the complex absolute value.

21906 **ERRORS**

21907 No errors are defined.

21908 **EXAMPLES**

21909 None.

21910 **APPLICATION USAGE**

21911 None.

21912 **RATIONALE**

21913 None.

21914 **FUTURE DIRECTIONS**

21915 None.

21916 **SEE ALSO**

21917 XBD <complex.h>

21918 **CHANGE HISTORY**

21919 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21920 **NAME**
 21921 cacos, cacosf, cacosl \mp complex arc cosine functions

21922 **SYNOPSIS**
 21923 #include <complex.h>
 21924 double complex cacos(double complex z);
 21925 float complex cacosf(float complex z);
 21926 long double complex cacosl(long double complex z);

21927 **DESCRIPTION**
 21928 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21929 conflict between the requirements described here and the ISO C standard is unintentional. This
 21930 volume of POSIX.1-2017 defers to the ISO C standard.

21931 These functions shall compute the complex arc cosine of z , with branch cuts outside the interval
 21932 $[-1, +1]$ along the real axis.

21933 **RETURN VALUE**
 21934 These functions shall return the complex arc cosine value, in the range of a strip mathematically
 21935 unbounded along the imaginary axis and in the interval $[0, \pi]$ along the real axis.

21936 **ERRORS**
 21937 No errors are defined.

21938 **EXAMPLES**
 21939 None.

21940 **APPLICATION USAGE**
 21941 None.

21942 **RATIONALE**
 21943 None.

21944 **FUTURE DIRECTIONS**
 21945 None.

21946 **SEE ALSO**
 21947 [ccos\(\)](#)
 21948 XBD [<complex.h>](#)

21949 **CHANGE HISTORY**
 21950 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

21951 **NAME**21952 cacosh, cacoshf, cacoshl \dagger complex arc hyperbolic cosine functions21953 **SYNOPSIS**

21954 #include <complex.h>

21955 double complex cacosh(double complex z);

21956 float complex cacoshf(float complex z);

21957 long double complex cacoshl(long double complex z);

21958 **DESCRIPTION**21959 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21960 conflict between the requirements described here and the ISO C standard is unintentional. This
21961 volume of POSIX.1-2017 defers to the ISO C standard.21962 These functions shall compute the complex arc hyperbolic cosine of z , with a branch cut at
21963 values less than 1 along the real axis.21964 **RETURN VALUE**21965 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip
21966 of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.21967 **ERRORS**

21968 No errors are defined.

21969 **EXAMPLES**

21970 None.

21971 **APPLICATION USAGE**

21972 None.

21973 **RATIONALE**

21974 None.

21975 **FUTURE DIRECTIONS**

21976 None.

21977 **SEE ALSO**21978 [ccosh\(\)](#)21979 XBD [<complex.h>](#)21980 **CHANGE HISTORY**

21981 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21982 **NAME**

21983 cacosl †'complex ar cosine functions

21984 **SYNOPSIS**

21985 #include <complex.h>

21986 long double complex cacosl(long double complex z);

21987 **DESCRIPTION**21988 Refer to *cacos()*.

21989 **NAME**21990 calloc \ddagger 'a memory allocator21991 **SYNOPSIS**

21992 #include <stdlib.h>

21993 void *calloc(size_t *nelem*, size_t *elsize*);21994 **DESCRIPTION**

21995 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21996 conflict between the requirements described here and the ISO C standard is unintentional. This
 21997 volume of POSIX.1-2017 defers to the ISO C standard.

21998 The *calloc()* function shall allocate unused space for an array of *nelem* elements each of whose
 21999 size in bytes is *elsize*. The space shall be initialized to all bits 0.

22000 The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The
 22001 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 22002 a pointer to any type of object and then used to access such an object or an array of such objects
 22003 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall
 22004 yield a pointer to an object disjoint from any other object. The pointer returned shall point to the
 22005 start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer
 22006 shall be returned. If the size of the space requested is 0, the behavior is implementation-defined:
 22007 either a null pointer shall be returned, or the behavior shall be as if the size were some non-zero
 22008 value, except that the behavior is undefined if the returned pointer is used to access an object.

22009 **RETURN VALUE**

22010 Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to
 22011 the allocated space. If either *nelem* or *elsize* is 0, then either:

22012 CX A null pointer shall be returned and *errno* may be set to an implementation-defined value,
 22013 or

22014 A pointer to the allocated space shall be returned. The application shall ensure that the
 22015 pointer is not used to access an object.

22016 CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

22017 **ERRORS**

22018 The *calloc()* function shall fail if:

22019 CX [ENOMEM] Insufficient memory is available.

22020 **EXAMPLES**

22021 None.

22022 **APPLICATION USAGE**

22023 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

22024 **RATIONALE**

22025 None.

22026 **FUTURE DIRECTIONS**

22027 None.

22028 **SEE ALSO**

22029 *free()*, *malloc()*, *realloc()*

22030 XBD <stdlib.h>

22031 **CHANGE HISTORY**

22032 First released in Issue 1. Derived from Issue 1 of the SVID.

22033 **Issue 6**

22034 Extensions beyond the ISO C standard are marked.

22035 The following new requirements on POSIX implementations derive from alignment with the
22036 Single UNIX Specification:

22037 The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient
22038 memory condition occurs.

22039 **Issue 7**

22040 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0053 [526] is applied.

22041 **NAME**

22042 carg, cargf, cargl — complex argument functions

22043 **SYNOPSIS**

22044 #include <complex.h>

22045 double carg(double complex z);

22046 float cargf(float complex z);

22047 long double cargl(long double complex z);

22048 **DESCRIPTION**22049 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
22050 conflict between the requirements described here and the ISO C standard is unintentional. This
22051 volume of POSIX.1-2017 defers to the ISO C standard.22052 These functions shall compute the argument (also called phase angle) of z , with a branch cut
22053 along the negative real axis.22054 **RETURN VALUE**22055 These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.22056 **ERRORS**

22057 No errors are defined.

22058 **EXAMPLES**

22059 None.

22060 **APPLICATION USAGE**

22061 None.

22062 **RATIONALE**

22063 None.

22064 **FUTURE DIRECTIONS**

22065 None.

22066 **SEE ALSO**22067 *cimag()*, *conj()*, *cproj()*

22068 XBD <complex.h>

22069 **CHANGE HISTORY**

22070 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22071 **NAME**22072 casin, casinf, casinl \mp 'complex arc sine functions22073 **SYNOPSIS**

22074 #include <complex.h>

22075 double complex casin(double complex z);

22076 float complex casinf(float complex z);

22077 long double complex casinl(long double complex z);

22078 **DESCRIPTION**

22079 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22080 conflict between the requirements described here and the ISO C standard is unintentional. This
 22081 volume of POSIX.1-2017 defers to the ISO C standard.

22082 These functions shall compute the complex arc sine of z , with branch cuts outside the interval
 22083 $[-1, +1]$ along the real axis.

22084 **RETURN VALUE**

22085 These functions shall return the complex arc sine value, in the range of a strip mathematically
 22086 unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

22087 **ERRORS**

22088 No errors are defined.

22089 **EXAMPLES**

22090 None.

22091 **APPLICATION USAGE**

22092 None.

22093 **RATIONALE**

22094 None.

22095 **FUTURE DIRECTIONS**

22096 None.

22097 **SEE ALSO**22098 [csin\(\)](#)22099 XBD [<complex.h>](#)22100 **CHANGE HISTORY**

22101 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22102 **NAME**

22103 casinh, casinhf, casinhl ‡complex arc hyperbolic sine functions

22104 **SYNOPSIS**

22105 #include <complex.h>

22106 double complex casinh(double complex z);

22107 float complex casinhf(float complex z);

22108 long double complex casinhl(long double complex z);

22109 **DESCRIPTION**

22110 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22111 conflict between the requirements described here and the ISO C standard is unintentional. This
22112 volume of POSIX.1-2017 defers to the ISO C standard.

22113 These functions shall compute the complex arc hyperbolic sine of z , with branch cuts outside the
22114 interval $[-i, +i]$ along the imaginary axis.

22115 **RETURN VALUE**

22116 These functions shall return the complex arc hyperbolic sine value, in the range of a strip
22117 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
22118 imaginary axis.

22119 **ERRORS**

22120 No errors are defined.

22121 **EXAMPLES**

22122 None.

22123 **APPLICATION USAGE**

22124 None.

22125 **RATIONALE**

22126 None.

22127 **FUTURE DIRECTIONS**

22128 None.

22129 **SEE ALSO**22130 [csinh\(\)](#)22131 XBD [<complex.h>](#)22132 **CHANGE HISTORY**

22133 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22134 **NAME**22135 casinl \dagger 'complex and sine functions22136 **SYNOPSIS**

22137 #include <complex.h>

22138 long double complex casinl(long double complex z);

22139 **DESCRIPTION**22140 Refer to *casin()*.

22141 **NAME**

22142 catan, catanf, catanl ‡complex arc tangent functions

22143 **SYNOPSIS**

22144 #include <complex.h>

22145 double complex catan(double complex z);

22146 float complex catanf(float complex z);

22147 long double complex catanl(long double complex z);

22148 **DESCRIPTION**

22149 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22150 conflict between the requirements described here and the ISO C standard is unintentional. This
22151 volume of POSIX.1-2017 defers to the ISO C standard.

22152 These functions shall compute the complex arc tangent of z , with branch cuts outside the
22153 interval $[-i, +i]$ along the imaginary axis.

22154 **RETURN VALUE**

22155 These functions shall return the complex arc tangent value, in the range of a strip
22156 mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the
22157 real axis.

22158 **ERRORS**

22159 No errors are defined.

22160 **EXAMPLES**

22161 None.

22162 **APPLICATION USAGE**

22163 None.

22164 **RATIONALE**

22165 None.

22166 **FUTURE DIRECTIONS**

22167 None.

22168 **SEE ALSO**22169 [ctan\(\)](#)22170 XBD [<complex.h>](#)22171 **CHANGE HISTORY**

22172 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22173 **NAME**22174 catanh, catanhf, catanhl \mp complex arc hyperbolic tangent functions22175 **SYNOPSIS**

22176 #include <complex.h>

22177 double complex catanh(double complex z);

22178 float complex catanhf(float complex z);

22179 long double complex catanhl(long double complex z);

22180 **DESCRIPTION**

22181 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22182 conflict between the requirements described here and the ISO C standard is unintentional. This
 22183 volume of POSIX.1-2017 defers to the ISO C standard.

22184 These functions shall compute the complex arc hyperbolic tangent of z , with branch cuts outside
 22185 the interval $[-1, +1]$ along the real axis.

22186 **RETURN VALUE**

22187 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip
 22188 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
 22189 imaginary axis.

22190 **ERRORS**

22191 No errors are defined.

22192 **EXAMPLES**

22193 None.

22194 **APPLICATION USAGE**

22195 None.

22196 **RATIONALE**

22197 None.

22198 **FUTURE DIRECTIONS**

22199 None.

22200 **SEE ALSO**22201 [ctanh\(\)](#)22202 XBD [<complex.h>](#)22203 **CHANGE HISTORY**

22204 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22205 **NAME**

22206 catanl †'complex ar tangent functions

22207 **SYNOPSIS**

22208 #include <complex.h>

22209 long double complex catanl(long double complex z);

22210 **DESCRIPTION**

22211 Refer to *catan()*.

22212 **NAME**

22213 catclose †'close a message catalog descriptor

22214 **SYNOPSIS**

22215 #include <nl_types.h>

22216 int catclose(nl_catd catd);

22217 **DESCRIPTION**22218 The *catclose()* function shall close the message catalog identified by *catd*. If a file descriptor is
22219 used to implement the type **nl_catd**, that file descriptor shall be closed.22220 **RETURN VALUE**22221 Upon successful completion, *catclose()* shall return 0; otherwise, -1 shall be returned, and *errno*
22222 set to indicate the error.22223 **ERRORS**22224 The *catclose()* function may fail if:

22225 [EBADF] The catalog descriptor is not valid.

22226 [EINTR] The *catclose()* function was interrupted by a signal.22227 **EXAMPLES**

22228 None.

22229 **APPLICATION USAGE**

22230 None.

22231 **RATIONALE**

22232 None.

22233 **FUTURE DIRECTIONS**

22234 None.

22235 **SEE ALSO**22236 *catgets()*, *catopen()*

22237 XBD <nl_types.h>

22238 **CHANGE HISTORY**

22239 First released in Issue 2.

22240 **Issue 7**22241 The *catclose()* function is moved from the XSI option to the Base.

22242 **NAME**

22243 catgets — read a program message

22244 **SYNOPSIS**

22245 #include <nl_types.h>

22246 char *catgets(nl_catd *catd*, int *set_id*, int *msg_id*, const char **s*);22247 **DESCRIPTION**

22248 The *catgets()* function shall attempt to read message *msg_id*, in set *set_id*, from the message
 22249 catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an
 22250 earlier call to *catopen()*. The results are undefined if *catd* is not a value returned by *catopen()* for
 22251 a message catalog still open in the process. The *s* argument points to a default message string
 22252 which shall be returned by *catgets()* if it cannot retrieve the identified message.

22253 The *catgets()* function need not be thread-safe.22254 **RETURN VALUE**

22255 If the identified message is retrieved successfully, *catgets()* shall return a pointer to an internal
 22256 buffer area containing the null-terminated message string. If the call is unsuccessful for any
 22257 reason, *s* shall be returned and *errno* shall be set to indicate the error.

22258 **ERRORS**22259 The *catgets()* function shall fail if:

22260 [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 22261 was transferred.

22262 [ENOMSG] The message identified by *set_id* and *msg_id* is not in the message catalog.

22263 The *catgets()* function may fail if:

22264 [EBADF] The *catd* argument is not a valid message catalog descriptor open for reading.

22265 [EBADMSG] The message identified by *set_id* and *msg_id* in the specified message catalog
 22266 did not satisfy implementation-defined security criteria.

22267 [EINVAL] The message catalog identified by *catd* is corrupted.

22268 **EXAMPLES**

22269 None.

22270 **APPLICATION USAGE**

22271 None.

22272 **RATIONALE**

22273 None.

22274 **FUTURE DIRECTIONS**

22275 None.

22276 **SEE ALSO**22277 *catclose()*, *catopen()*

22278 XBD <nl_types.h>

22279 **CHANGE HISTORY**

22280 First released in Issue 2.

22281 **Issue 5**

22282 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

22283 **Issue 6**

22284 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

22285 **Issue 7**

22286 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [EINTR] and [ENOMSG] errors to become “shall fail” errors, updating the RETURN VALUE section, and updating the DESCRIPTION to note that: “The results are undefined if *catd* is not a value returned by *catopen()* for a message catalog still open in the process.

22290 Austin Group Interpretation 1003.1-2001 #148 is applied, adding

22291 The *catgets()* function is moved from the XSI option to the Base.

22292 **NAME**22293 `catopen` ‡open a message catalog22294 **SYNOPSIS**22295 `#include <nl_types.h>`22296 `nl_catd catopen(const char *name, int oflag);`22297 **DESCRIPTION**

22298 The `catopen()` function shall open a message catalog and return a message catalog descriptor.
 22299 The `name` argument specifies the name of the message catalog to be opened. If `name` contains a
 22300 `'/'`, then `name` specifies a pathname for the message catalog. Otherwise, the environment
 22301 variable `NLSPATH` is used with `name` substituted for the `%N` conversion specification (see XBD
 22302 [Chapter 8](#), on page 173); if `NLSPATH` exists in the environment when the process starts, then if
 22303 the process has appropriate privileges, the behavior of `catopen()` is undefined. If `NLSPATH` does
 22304 not exist in the environment, or if a message catalog cannot be found in any of the components
 22305 specified by `NLSPATH`, then an implementation-defined default path shall be used. This default
 22306 may be affected by the setting of `LC_MESSAGES` if the value of `oflag` is `NL_CAT_LOCALE`, or
 22307 the `LANG` environment variable if `oflag` is 0.

22308 A message catalog descriptor shall remain valid in a process until that process closes it, or a
 22309 successful call to one of the `exec` functions. A change in the setting of the `LC_MESSAGES`
 22310 category may invalidate existing open catalogs.

22311 If a file descriptor is used to implement message catalog descriptors, the `FD_CLOEXEC` flag
 22312 shall be set; see `<fcntl.h>`.

22313 If the value of the `oflag` argument is 0, the `LANG` environment variable is used to locate the
 22314 catalog without regard to the `LC_MESSAGES` category. If the `oflag` argument is
 22315 `NL_CAT_LOCALE`, the `LC_MESSAGES` category is used to locate the message catalog (see XBD
 22316 [Section 8.2](#), on page 174).

22317 **RETURN VALUE**

22318 Upon successful completion, `catopen()` shall return a message catalog descriptor for use on
 22319 subsequent calls to `catgets()` and `catclose()`. Otherwise, `catopen()` shall return `(nl_catd) -1` and set
 22320 `errno` to indicate the error.

22321 **ERRORS**22322 The `catopen()` function may fail if:

22323 [EACCES] Search permission is denied for the component of the path prefix of the
 22324 message catalog or read permission is denied for the message catalog.

22325 [EMFILE] All file descriptors available to the process are currently open.

22326 [ENAMETOOLONG]

22327 The length of a component of a pathname is longer than `{NAME_MAX}`.

22328 [ENAMETOOLONG]

22329 The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a
 22330 symbolic link produced an intermediate result with a length that exceeds
 22331 `{PATH_MAX}`.

22332 [ENFILE] Too many files are currently open in the system.

22333 [ENOENT] The message catalog does not exist or the `name` argument points to an empty
 22334 string.

22335 [ENOMEM] Insufficient storage space is available.

22336 [ENOTDIR] A component of the path prefix of the message catalog names an existing file
22337 that is neither a directory nor a symbolic link to a directory, or the pathname
22338 of the message catalog contains at least one non-`<slash>` character and ends
22339 with one or more trailing `<slash>` characters and the last pathname
22340 component names an existing file that is neither a directory nor a symbolic
22341 link to a directory.

EXAMPLES

22342 None.
22343

APPLICATION USAGE

22344 Some implementations of `catopen()` use `malloc()` to allocate space for internal buffer areas. The
22345 `catopen()` function may fail if there is insufficient storage space available to accommodate these
22346 buffers.
22347

22348 Conforming applications must assume that message catalog descriptors are not valid after a call
22349 to one of the `exec` functions.

22350 Application developers should be aware that guidelines for the location of message catalogs
22351 have not yet been developed. Therefore they should take care to avoid conflicting with catalogs
22352 used by other applications and the standard utilities.

22353 To be sure that messages produced by an application running with appropriate privileges cannot
22354 be used by an attacker setting an unexpected value for `NLSPATH` in the environment to confuse
22355 a system administrator, such applications should use pathnames containing a `'/'` to get defined
22356 behavior when using `catopen()` to open a message catalog.

RATIONALE

22357 None.
22358

FUTURE DIRECTIONS

22359 None.
22360

SEE ALSO

22361 [catclose\(\)](#), [catgets\(\)](#)

22362 XBD Chapter 8 (on page 173), [<fcntl.h>](#), [<nl_types.h>](#),

CHANGE HISTORY

22364 First released in Issue 2.
22365

Issue 7

22366 Austin Group Interpretation 1003.1-2001 #143 is applied.

22367 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

22368 The `catopen()` function is moved from the XSI option to the Base.

22369 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0045 [324] is applied.

22370 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0054 [645], XSH/TC2-2008/0055 [497],
22371 and XSH/TC2-2008/0056 [497] are applied.
22372

22373 **NAME**

22374 cbrt, cbrtf, cbrtl — cube root functions

22375 **SYNOPSIS**

```
22376 #include <math.h>
22377 double cbrt(double x);
22378 float cbrtf(float x);
22379 long double cbrtl(long double x);
```

22380 **DESCRIPTION**

22381 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22382 conflict between the requirements described here and the ISO C standard is unintentional. This
 22383 volume of POSIX.1-2017 defers to the ISO C standard.

22384 These functions shall compute the real cube root of their argument x .22385 **RETURN VALUE**22386 Upon successful completion, these functions shall return the cube root of x .22387 MX If x is NaN, a NaN shall be returned.22388 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.22389 **ERRORS**

22390 No errors are defined.

22391 **EXAMPLES**

22392 None.

22393 **APPLICATION USAGE**

22394 None.

22395 **RATIONALE**

22396 For some applications, a true cube root function, which returns negative results for negative
 22397 arguments, is more appropriate than $\text{pow}(x, 1.0/3.0)$, which returns a NaN for x less than 0.

22398 **FUTURE DIRECTIONS**

22399 None.

22400 **SEE ALSO**22401 XBD [<math.h>](#)22402 **CHANGE HISTORY**

22403 First released in Issue 4, Version 2.

22404 **Issue 5**

22405 Moved from X/OPEN UNIX extension to BASE.

22406 **Issue 6**22407 The `cbrt()` function is no longer marked as an extension.22408 The `cbrtf()` and `cbrtl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

22409 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 22410 revised to align with the ISO/IEC 9899:1999 standard.

22411 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 22412 marked.

22413 **NAME**

22414 ccos, ccosf, ccosl ‡complex cosine functions

22415 **SYNOPSIS**

22416 #include <complex.h>

22417 double complex ccos(double complex z);

22418 float complex ccosf(float complex z);

22419 long double complex ccosl(long double complex z);

22420 **DESCRIPTION**

22421 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22422 conflict between the requirements described here and the ISO C standard is unintentional. This
22423 volume of POSIX.1-2017 defers to the ISO C standard.

22424 These functions shall compute the complex cosine of z.

22425 **RETURN VALUE**

22426 These functions shall return the complex cosine value.

22427 **ERRORS**

22428 No errors are defined.

22429 **EXAMPLES**

22430 None.

22431 **APPLICATION USAGE**

22432 None.

22433 **RATIONALE**

22434 None.

22435 **FUTURE DIRECTIONS**

22436 None.

22437 **SEE ALSO**22438 [ccos\(\)](#)22439 XBD [<complex.h>](#)22440 **CHANGE HISTORY**

22441 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22442 **NAME**22443 `ccosh, ccoshf, ccoshl` ‡complex hyperbolic cosine functions22444 **SYNOPSIS**22445 `#include <complex.h>`22446 `double complex ccosh(double complex z);`22447 `float complex ccoshf(float complex z);`22448 `long double complex ccoshl(long double complex z);`22449 **DESCRIPTION**

22450 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22451 conflict between the requirements described here and the ISO C standard is unintentional. This
22452 volume of POSIX.1-2017 defers to the ISO C standard.

22453 These functions shall compute the complex hyperbolic cosine of z .22454 **RETURN VALUE**

22455 These functions shall return the complex hyperbolic cosine value.

22456 **ERRORS**

22457 No errors are defined.

22458 **EXAMPLES**

22459 None.

22460 **APPLICATION USAGE**

22461 None.

22462 **RATIONALE**

22463 None.

22464 **FUTURE DIRECTIONS**

22465 None.

22466 **SEE ALSO**22467 [cacosh\(\)](#)22468 XBD [<complex.h>](#)22469 **CHANGE HISTORY**

22470 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22471 **NAME**

22472 ccosl ‡'complex cosine functions

22473 **SYNOPSIS**

22474 #include <complex.h>

22475 long double complex ccosl(long double complex z);

22476 **DESCRIPTION**22477 Refer to *ccos()*.

22478 **NAME**

22479 ceil, ceilf, ceill ‡'ceiling value function

22480 **SYNOPSIS**

```
22481       #include <math.h>
22482       double ceil(double x);
22483       float ceilf(float x);
22484       long double ceill(long double x);
```

22485 **DESCRIPTION**

22486 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22487 conflict between the requirements described here and the ISO C standard is unintentional. This
 22488 volume of POSIX.1-2017 defers to the ISO C standard.

22489 These functions shall compute the smallest integral value not less than x .

22490 **RETURN VALUE**

22491 MX The result shall have the same sign as x .

22492 Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not
 22493 less than x , expressed as a type **double**, **float**, or **long double**, respectively.

22494 MX If x is NaN, a NaN shall be returned.

22495 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

22496 **ERRORS**

22497 No errors are defined.

22498 **EXAMPLES**

22499 None.

22500 **APPLICATION USAGE**

22501 The integral value returned by these functions need not be expressible as an **intmax_t**. The
 22502 return value should be tested before assigning it to an integer type to avoid the undefined
 22503 results of an integer overflow.

22504 These functions may raise the inexact floating-point exception if the result differs in value from
 22505 the argument.

22506 **RATIONALE**

22507 None.

22508 **FUTURE DIRECTIONS**

22509 None.

22510 **SEE ALSO**

22511 *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*

22512 XBD Section 4.20 (on page 117), **<math.h>**

22513 **CHANGE HISTORY**

22514 First released in Issue 1. Derived from Issue 1 of the SVID.

22515 **Issue 5**

22516 The DESCRIPTION is updated to indicate how an application should check for an error. This
 22517 text was previously published in the APPLICATION USAGE section.

22518 **Issue 6**22519 The *ceilf()* and *ceil()* functions are added for alignment with the ISO/IEC 9899:1999 standard.22520 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
22521 revised to align with the ISO/IEC 9899:1999 standard.22522 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
22523 marked.22524 **Issue 7**

22525 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0046 [346] is applied.

22526 **NAME**

22527 cexp, cexpf, cexpl ‡complex exponential functions

22528 **SYNOPSIS**

22529 #include <complex.h>

22530 double complex cexp(double complex z);

22531 float complex cexpf(float complex z);

22532 long double complex cexpl(long double complex z);

22533 **DESCRIPTION**

22534 CX The functionality described on this reference page is aligned with the ISO C standard. Any
22535 conflict between the requirements described here and the ISO C standard is unintentional. This
22536 volume of POSIX.1-2017 defers to the ISO C standard.

22537 These functions shall compute the complex exponent of z , defined as e^z .22538 **RETURN VALUE**22539 These functions shall return the complex exponential value of z .22540 **ERRORS**

22541 No errors are defined.

22542 **EXAMPLES**

22543 None.

22544 **APPLICATION USAGE**

22545 None.

22546 **RATIONALE**

22547 None.

22548 **FUTURE DIRECTIONS**

22549 None.

22550 **SEE ALSO**22551 [clog\(\)](#)22552 XBD [<complex.h>](#)22553 **CHANGE HISTORY**

22554 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22555 **NAME**

22556 cfgetispeed ¶get input baud rate

22557 **SYNOPSIS**

22558 #include <termios.h>

22559 speed_t cfgetispeed(const struct termios *termios_p);

22560 **DESCRIPTION**22561 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which
22562 the *termios_p* argument points.22563 This function shall return exactly the value in the **termios** data structure, without interpretation.22564 **RETURN VALUE**22565 Upon successful completion, *cfgetispeed()* shall return a value of type **speed_t** representing the
22566 input baud rate.22567 **ERRORS**

22568 No errors are defined.

22569 **EXAMPLES**

22570 None.

22571 **APPLICATION USAGE**

22572 None.

22573 **RATIONALE**22574 The term “baud” is used historically here, but is not technically correct. This is properly “bits per
22575 second”, which may not be the same as baud. However, the term is used because of the
22576 historical usage and understanding.22577 The *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as
22578 numbers, but rather as symbolic names. There are two reasons for this:

- 22579 1. Historically, numbers were not used because of the way the rate was stored in the data
-
- 22580 structure. This is retained even though a function is now used.
-
- 22581 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
-
- 22582 the application to that set.

22583 There is nothing to prevent an implementation accepting as an extension a number (such as 126),
22584 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid
22585 introducing ambiguity.22586 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications
22587 in this volume of POSIX.1-2017 have made it possible to determine whether split rates are
22588 supported and to support them without having to treat zero as a special case. Since this
22589 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is
22590 the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-2017 does not
22591 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit
22592 from the two being equivalent. This volume of POSIX.1-2017 does not fully specify whether the
22593 previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as zero. Therefore,
22594 conforming applications should always set both the input speed and output speed when setting
22595 either.22596 In historical implementations, the baud rate information is traditionally kept in **c_cflag**.
22597 Applications should be written to presume that this might be the case (and thus not blindly copy
22598 **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag**
22599 field absolutely after setting a baud rate is a non-portable action because of this. In general, the

22600 unused parts of the flag fields might be used by the implementation and should not be blindly
22601 copied from the descriptions of one terminal device to another.

22602 **FUTURE DIRECTIONS**

22603 None.

22604 **SEE ALSO**

22605 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*

22606 XBD Chapter 11 (on page 199), [<termios.h>](#)

22607 **CHANGE HISTORY**

22608 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22609 **NAME**

22610 cfgetospeed ¶get output baud rate

22611 **SYNOPSIS**

22612 #include <termios.h>

22613 speed_t cfgetospeed(const struct termios *termios_p);

22614 **DESCRIPTION**

22615 The *cfgetospeed()* function shall extract the output baud rate from the **termios** structure to which
22616 the *termios_p* argument points.

22617 This function shall return exactly the value in the **termios** data structure, without interpretation.

22618 **RETURN VALUE**

22619 Upon successful completion, *cfgetospeed()* shall return a value of type **speed_t** representing the
22620 output baud rate.

22621 **ERRORS**

22622 No errors are defined.

22623 **EXAMPLES**

22624 None.

22625 **APPLICATION USAGE**

22626 None.

22627 **RATIONALE**

22628 Refer to *cfgetispeed()*.

22629 **FUTURE DIRECTIONS**

22630 None.

22631 **SEE ALSO**

22632 *cfgetispeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*

22633 XBD Chapter 11 (on page 199), <**termios.h**>

22634 **CHANGE HISTORY**

22635 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22636 **NAME**

22637 cfsetispeed ¶set input baud rate

22638 **SYNOPSIS**

22639 #include <termios.h>

22640 int cfsetispeed(struct termios *termios_p, speed_t speed);

22641 **DESCRIPTION**22642 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by
22643 *termios_p* to *speed*.22644 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
22645 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
22646 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
22647 function is called.22648 **RETURN VALUE**22649 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and
22650 *errno* may be set to indicate the error.22651 **ERRORS**22652 The *cfsetispeed()* function may fail if:22653 [EINVAL] The *speed* value is not a valid baud rate.22654 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
22655 <termios.h>.22656 **EXAMPLES**

22657 None.

22658 **APPLICATION USAGE**

22659 None.

22660 **RATIONALE**22661 Refer to *cfgetispeed()*.22662 **FUTURE DIRECTIONS**

22663 None.

22664 **SEE ALSO**22665 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*

22666 XBD Chapter 11 (on page 199), <termios.h>

22667 **CHANGE HISTORY**

22668 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22669 **Issue 6**22670 The following new requirements on POSIX implementations derive from alignment with the
22671 Single UNIX Specification:22672 The optional setting of *errno* and the [EINVAL] error conditions are added.

22673 **NAME**

22674 cfsetospeed ¶set output baud rate

22675 **SYNOPSIS**

22676 #include <termios.h>

22677 int cfsetospeed(struct termios *termios_p, speed_t speed);

22678 **DESCRIPTION**22679 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by
22680 *termios_p* to *speed*.22681 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
22682 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
22683 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
22684 function is called.22685 **RETURN VALUE**22686 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*
22687 may be set to indicate the error.22688 **ERRORS**22689 The *cfsetospeed()* function may fail if:22690 [EINVAL] The *speed* value is not a valid baud rate.22691 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
22692 <termios.h>.22693 **EXAMPLES**

22694 None.

22695 **APPLICATION USAGE**

22696 None.

22697 **RATIONALE**22698 Refer to *cfgetispeed()*.22699 **FUTURE DIRECTIONS**

22700 None.

22701 **SEE ALSO**22702 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*

22703 XBD Chapter 11 (on page 199), <termios.h>

22704 **CHANGE HISTORY**

22705 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

22706 **Issue 6**22707 The following new requirements on POSIX implementations derive from alignment with the
22708 Single UNIX Specification:22709 The optional setting of *errno* and the [EINVAL] error conditions are added.

22710 **NAME**

22711 chdir — change working directory

22712 **SYNOPSIS**

22713 #include <unistd.h>

22714 int chdir(const char *path);

22715 **DESCRIPTION**

22716 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*
 22717 argument to become the current working directory; that is, the starting point for path searches
 22718 for pathnames not beginning with '/ '.

22719 **RETURN VALUE**

22720 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current
 22721 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

22722 **ERRORS**22723 The *chdir()* function shall fail if:

22724 [EACCES] Search permission is denied for any component of the pathname.

22725 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 22726 argument.

22727 [ENAMETOOLONG]

22728 The length of a component of a pathname is longer than {NAME_MAX}.

22729 [ENOENT] A component of *path* does not name an existing directory or *path* is an empty
 22730 string.

22731 [ENOTDIR] A component of the pathname names an existing file that is neither a directory
 22732 nor a symbolic link to a directory.

22733 The *chdir()* function may fail if:

22734 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 22735 resolution of the *path* argument.

22736 [ENAMETOOLONG]

22737 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 22738 symbolic link produced an intermediate result with a length that exceeds
 22739 {PATH_MAX}.

22740 **EXAMPLES**22741 **Changing the Current Working Directory**

22742 The following example makes the value pointed to by **directory**, */tmp*, the current working
 22743 directory.

22744 #include <unistd.h>

22745 ...

22746 char *directory = "/tmp";

22747 int ret;

22748 ret = chdir (directory);

22749 **APPLICATION USAGE**

22750 None.

22751 **RATIONALE**22752 The *chdir()* function only affects the working directory of the current process.22753 **FUTURE DIRECTIONS**

22754 None.

22755 **SEE ALSO**22756 *getcwd()*22757 XBD <*unistd.h*>22758 **CHANGE HISTORY**

22759 First released in Issue 1. Derived from Issue 1 of the SVID.

22760 **Issue 6**

22761 The APPLICATION USAGE section is added.

22762 The following new requirements on POSIX implementations derive from alignment with the
22763 Single UNIX Specification:

22764 The [ELOOP] mandatory error condition is added.

22765 A second [ENAMETOOLONG] is added as an optional error condition.

22766 The following changes were made to align with the IEEE P1003.1a draft standard:

22767 The [ELOOP] optional error condition is added.

22768 **Issue 7**

22769 Austin Group Interpretation 1003.1-2001 #143 is applied.

22770 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0047 [324] is applied.

22771 **NAME**

22772 chmod, fchmodat ‡change mode of a file

22773 **SYNOPSIS**

22774 #include <sys/stat.h>

22775 int chmod(const char *path, mode_t mode);

22776 OH #include <fcntl.h>

22777 int fchmodat(int fd, const char *path, mode_t mode, int flag);

22778 **DESCRIPTION**

22779 XSI The *chmod()* function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of
 22780 the file named by the pathname pointed to by the *path* argument to the corresponding bits in the
 22781 *mode* argument. The application shall ensure that the effective user ID of the process matches the
 22782 owner of the file or the process has appropriate privileges in order to do this.

22783 XSI S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in <sys/stat.h>.

22784 If the calling process does not have appropriate privileges, and if the group ID of the file does
 22785 not match the effective group ID or one of the supplementary group IDs and if the file is a
 22786 regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon
 22787 successful return from *chmod()*.

22788 Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in
 22789 *mode* to be ignored.

22790 Upon successful completion, *chmod()* shall mark for update the last file status change timestamp
 22791 of the file.

22792 The *fchmodat()* function shall be equivalent to the *chmod()* function except in the case where *path*
 22793 specifies a relative path. In this case the file to be changed is determined relative to the directory
 22794 associated with the file descriptor *fd* instead of the current working directory. If the access mode
 22795 of the open file description associated with the file descriptor is not O_SEARCH, the function
 22796 shall check whether directory searches are permitted using the current permissions of the
 22797 directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not
 22798 perform the check.

22799 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 22800 in <fcntl.h>:

22801 AT_SYMLINK_NOFOLLOW

22802 If *path* names a symbolic link, then the mode of the symbolic link is changed.

22803 If *fchmodat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 22804 directory shall be used. If also *flag* is zero, the behavior shall be identical to a call to *chmod()*.

22805 **RETURN VALUE**

22806 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 22807 return -1 and set *errno* to indicate the error. If -1 is returned, no change to the file mode occurs.

22808 **ERRORS**

22809 These functions shall fail if:

22810 [EACCES] Search permission is denied on a component of the path prefix.

22811 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 22812 argument.

22813	[ENAMETOOLONG]	
22814		The length of a component of a pathname is longer than {NAME_MAX}.
22815	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
22816	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>path</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
22817		
22818		
22819		
22820		
22821	[EPERM]	The effective user ID does not match the owner of the file and the process does not have appropriate privileges.
22822		
22823	[EROFS]	The named file resides on a read-only file system.
22824		The <i>fchmodat()</i> function shall fail if:
22825	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
22826		
22827		
22828	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
22829		
22830	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
22831		
22832		These functions may fail if:
22833	[EINTR]	A signal was caught during execution of the function.
22834	[EINVAL]	The value of the <i>mode</i> argument is invalid.
22835	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
22836		
22837	[ENAMETOOLONG]	
22838		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
22839		
22840		
22841		The <i>fchmodat()</i> function may fail if:
22842	[EINVAL]	The value of the <i>flag</i> argument is invalid.
22843	[EOPNOTSUPP]	The AT_SYMLINK_NOFOLLOW bit is set in the <i>flag</i> argument, <i>path</i> names a symbolic link, and the system does not support changing the mode of a symbolic link.
22844		
22845		

22846 **EXAMPLES**22847 **Setting Read Permissions for User, Group, and Others**

22848 The following example sets read permissions for the owner, group, and others.

```
22849 #include <sys/stat.h>
22850 const char *path;
22851 ...
22852 chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

22853 **Setting Read, Write, and Execute Permissions for the Owner Only**

22854 The following example sets read, write, and execute permissions for the owner, and no
22855 permissions for group and others.

```
22856 #include <sys/stat.h>
22857 const char *path;
22858 ...
22859 chmod(path, S_IRWXU);
```

22860 **Setting Different Permissions for Owner, Group, and Other**

22861 The following example sets owner permissions for CHANGEFILE to read, write, and execute,
22862 group permissions to read and execute, and other permissions to read.

```
22863 #include <sys/stat.h>
22864 #define CHANGEFILE "/etc/myfile"
22865 ...
22866 chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

22867 **Setting and Checking File Permissions**

22868 The following example sets the file permission bits for a file named `/home/cnd/mod1`, then calls
22869 the `stat()` function to verify the permissions.

```
22870 #include <sys/types.h>
22871 #include <sys/stat.h>
22872 int status;
22873 struct stat buffer
22874 ...
22875 chmod("/home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
22876 status = stat("/home/cnd/mod1", &buffer);
```

22877 **APPLICATION USAGE**

22878 In order to ensure that the `S_ISUID` and `S_ISGID` bits are set, an application requiring this
22879 should use `stat()` after a successful `chmod()` to verify this.

22880 Any file descriptors currently open by any process on the file could possibly become invalid if
22881 the mode of the file is changed to a value which would deny access to that process. One
22882 situation where this could occur is on a stateless file system. This behavior will not occur in a
22883 conforming environment.

22884 **RATIONALE**

22885 This volume of POSIX.1-2017 specifies that the S_ISGID bit is cleared by *chmod()* on a regular file
 22886 under certain conditions. This is specified on the assumption that regular files may be executed,
 22887 and the system should prevent users from making executable *setgid()* files perform with
 22888 privileges that the caller does not have. On implementations that support execution of other file
 22889 types, the S_ISGID bit should be cleared for those file types under the same circumstances.

22890 Implementations that use the S_ISUID bit to indicate some other function (for example,
 22891 mandatory record locking) on non-executable files need not clear this bit on writing. They
 22892 should clear the bit for executable files and any other cases where the bit grants special powers
 22893 to processes that change the file contents. Similar comments apply to the S_ISGID bit.

22894 The purpose of the *fchmodat()* function is to enable changing the mode of files in directories
 22895 other than the current working directory without exposure to race conditions. Any part of the
 22896 path of a file could be changed in parallel to a call to *chmod()*, resulting in unspecified behavior.
 22897 By opening a file descriptor for the target directory and using the *fchmodat()* function it can be
 22898 guaranteed that the changed file is located relative to the desired directory. Some
 22899 implementations might allow changing the mode of symbolic links. This is not supported by the
 22900 interfaces in the POSIX specification. Systems with such support provide an interface named
 22901 *lchmod()*. To support such implementations *fchmodat()* has a *flag* parameter.

22902 **FUTURE DIRECTIONS**

22903 None.

22904 **SEE ALSO**

22905 *access()*, *chown()*, *exec*, *fstatat()*, *fstatvfs()*, *mkdir()*, *mkfifo()*, *mknod()*, *open()*

22906 XBD [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)

22907 **CHANGE HISTORY**

22908 First released in Issue 1. Derived from Issue 1 of the SVID.

22909 **Issue 6**

22910 The following new requirements on POSIX implementations derive from alignment with the
 22911 Single UNIX Specification:

22912 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 22913 required for conforming implementations of previous POSIX specifications, it was not
 22914 required for UNIX applications.

22915 The [EINVAL] and [EINTR] optional error conditions are added.

22916 A second [ENAMETOOLONG] is added as an optional error condition.

22917 The following changes were made to align with the IEEE P1003.1a draft standard:

22918 The [ELOOP] optional error condition is added.

22919 The normative text is updated to avoid use of the term “must” for application requirements.

22920 **Issue 7**

22921 Austin Group Interpretation 1003.1-2001 #143 is applied.

22922 The *fchmodat()* function is added from The Open Group Technical Standard, 2006, Extended API
 22923 Set Part 2.

22924 Changes are made related to support for finegrained timestamps.

22925 Changes are made to allow a directory to be opened for searching.

22926 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a

22927 pathname exists but is not a directory or a symbolic link to a directory.

22928 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0048 [300], XSH/TC1-2008/0049 [461],
22929 XSH/TC1-2008/0050 [324], XSH/TC1-2008/0051 [278], and XSH/TC1-2008/0052 [278] are
22930 applied.

22931 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0057 [873], XSH/TC2-2008/0058 [591],
22932 XSH/TC2-2008/0059 [817], XSH/TC2-2008/0060 [817], and XSH/TC2-2008/0061 [893] are
22933 applied.

22934 **NAME**

22935 chown, fchownat — change owner and group of a file

22936 **SYNOPSIS**

22937 #include <unistd.h>

22938 int chown(const char *path, uid_t owner, gid_t group);

22939 OH #include <fcntl.h>

22940 int fchownat(int fd, const char *path, uid_t owner, gid_t group,

22941 int flag);

22942 **DESCRIPTION**22943 The *chown()* function shall change the user and group ownership of a file.22944 The *path* argument points to a pathname naming a file. The user ID and group ID of the named
22945 file shall be set to the numeric values contained in *owner* and *group*, respectively.22946 Only processes with an effective user ID equal to the user ID of the file or with appropriate
22947 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for
22948 *path*:

22949 Changing the user ID is restricted to processes with appropriate privileges.

22950 Changing the group ID is permitted to a process with an effective user ID equal to the user
22951 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's
22952 user ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to
22953 one of its supplementary group IDs.22954 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of
22955 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
22956 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful
22957 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,
22958 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is
22959 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()*
22960 function is successfully invoked on a file that is not a regular file and one or more of the
22961 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID
22962 bits may be cleared.22963 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the
22964 file shall not be changed.22965 Upon successful completion, *chown()* shall mark for update the last file status change timestamp
22966 of the file, except that if *owner* is `(uid_t)-1` and *group* is `(gid_t)-1`, the file status change
22967 timestamp need not be marked for update.22968 The *fchownat()* function shall be equivalent to the *chown()* and *lchown()* functions except in the
22969 case where *path* specifies a relative path. In this case the file to be changed is determined relative
22970 to the directory associated with the file descriptor *fd* instead of the current working directory. If
22971 the access mode of the open file description associated with the file descriptor is not
22972 `O_SEARCH`, the function shall check whether directory searches are permitted using the current
22973 permissions of the directory underlying the file descriptor. If the access mode is `O_SEARCH`, the
22974 function shall not perform the check.22975 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
22976 in `<fcntl.h>`:

22977 AT_SYMLINK_NOFOLLOW
 22978 If *path* names a symbolic link, ownership of the symbolic link is changed.

22979 If *fchownat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 22980 directory shall be used and the behavior shall be identical to a call to *chown()* or *lchown()*
 22981 respectively, depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in the *flag*
 22982 argument.

22983 **RETURN VALUE**
 22984 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 22985 return -1 and set *errno* to indicate the error. If -1 is returned, no changes are made in the user ID
 22986 and group ID of the file.

22987 **ERRORS**
 22988 These functions shall fail if:

22989 [EACCES] Search permission is denied on a component of the path prefix.

22990 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 22991 argument.

22992 [ENAMETOOLONG]
 22993 The length of a component of a pathname is longer than {NAME_MAX}.

22994 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

22995 [ENOTDIR] A component of the path prefix names an existing file that is neither a
 22996 directory nor a symbolic link to a directory, or the *path* argument contains at
 22997 least one non-`<slash>` character and ends with one or more trailing `<slash>`
 22998 characters and the last pathname component names an existing file that is
 22999 neither a directory nor a symbolic link to a directory.

23000 [EPERM] The effective user ID does not match the owner of the file, or the calling
 23001 process does not have appropriate privileges and
 23002 _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

23003 [EROFS] The named file resides on a read-only file system.

23004 The *fchownat()* function shall fail if:

23005 [EACCES] The access mode of the open file description associated with *fd* is not
 23006 O_SEARCH and the permissions of the directory underlying *fd* do not permit
 23007 directory searches.

23008 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 23009 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

23010 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
 23011 with a non-directory file.

23012 These functions may fail if:

23013 [EIO] An I/O error occurred while reading or writing to the file system.

23014 [EINTR] The *chown()* function was interrupted by a signal which was caught.

23015 [EINVAL] The owner or group ID supplied is not a value supported by the
 23016 implementation.

23017 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
23018 resolution of the *path* argument.

23019 [ENAMETOOLONG]
23020 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
23021 symbolic link produced an intermediate result with a length that exceeds
23022 {PATH_MAX}.

23023 The *fchownat()* function may fail if:

23024 [EINVAL] The value of the *flag* argument is not valid.

23025 EXAMPLES

23026 None.

23027 APPLICATION USAGE

23028 Although *chown()* can be used on some implementations by the file owner to change the owner
23029 and group to any desired values, the only portable use of this function is to change the group of
23030 a file to the effective GID of the calling process or to a member of its group set.

23031 RATIONALE

23032 System III and System V allow a user to give away files; that is, the owner of a file may change
23033 its user ID to anything. This is a serious problem for implementations that are intended to meet
23034 government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the
23035 user ID of a file. Some government agencies (usually not ones concerned directly with security)
23036 find this limitation too confining. This volume of POSIX.1-2017 uses *may* to permit secure
23037 implementations while not disallowing System V.

23038 System III and System V allow the owner of a file to change the group ID to anything. Version 7
23039 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to
23040 change the group ID of a file to its effective group ID or to any of the groups in the list of
23041 supplementary group IDs, but to no others.

23042 The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate
23043 privileged process clear the S_ISGID and the S_ISUID bits for regular files, and permits them to
23044 be cleared for other types of files. This is so that changes in accessibility do not accidentally
23045 cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-
23046 executable data files also clears the mandatory file locking bit (shared with S_ISGID), which is
23047 an extension on many implementations (it first appeared in System V). These bits should only be
23048 required to be cleared on regular files that have one or more of their execute bits set.

23049 The purpose of the *fchownat()* function is to enable changing ownership of files in directories
23050 other than the current working directory without exposure to race conditions. Any part of the
23051 path of a file could be changed in parallel to a call to *chown()* or *lchown()*, resulting in
23052 unspecified behavior. By opening a file descriptor for the target directory and using the
23053 *fchownat()* function it can be guaranteed that the changed file is located relative to the desired
23054 directory.

23055 FUTURE DIRECTIONS

23056 None.

23057 SEE ALSO

23058 [*chmod\(\)*](#), [*fpathconf\(\)*](#), [*lchown\(\)*](#)

23059 XBD [*<fcntl.h>*](#), [*<sys/types.h>*](#), [*<unistd.h>*](#)

23060 **CHANGE HISTORY**

23061 First released in Issue 1. Derived from Issue 1 of the SVID.

23062 **Issue 6**

23063 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

23064 The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is
23065 restored.23066 The `[EPERM]` error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`.
23067 This is since its operand is a pathname and applications should be aware that the error
23068 may not occur for that pathname if the file system does not support
23069 `_POSIX_CHOWN_RESTRICTED`.23070 The following new requirements on POSIX implementations derive from alignment with the
23071 Single UNIX Specification:23072 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
23073 required for conforming implementations of previous POSIX specifications, it was not
23074 required for UNIX applications.23075 The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the
23076 group ID only. A corresponding change is made for group.23077 The `[ELOOP]` mandatory error condition is added.23078 The `[EIO]` and `[EINTR]` optional error conditions are added.23079 A second `[ENAMETOOLONG]` is added as an optional error condition.

23080 The following changes were made to align with the IEEE P1003.1a draft standard:

23081 Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when
23082 the process has appropriate privileges.23083 The `[ELOOP]` optional error condition is added.23084 **Issue 7**

23085 Austin Group Interpretation 1003.1-2001 #143 is applied.

23086 The `fchownat()` function is added from The Open Group Technical Standard, 2006, Extended API
23087 Set Part 2.

23088 Changes are made related to support for finegrained timestamps.

23089 Changes are made to allow a directory to be opened for searching.

23090 The `[ENOTDIR]` error condition is clarified to cover the condition where the last component of a
23091 pathname exists but is not a directory or a symbolic link to a directory.23092 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0053 [461], XSH/TC1-2008/0054 [324],
23093 XSH/TC1-2008/0055 [278], and XSH/TC1-2008/0056 [278] are applied.23094 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0062 [873], XSH/TC2-2008/0063 [591],
23095 XSH/TC2-2008/0064 [485], XSH/TC2-2008/0065 [817], and XSH/TC2-2008/0066 [817] are
23096 applied.

23097 **NAME**

23098 cimag, cimagf, cimagl ‡complex imaginary functions

23099 **SYNOPSIS**

```
23100       #include <complex.h>
23101       double cimag(double complex z);
23102       float cimagf(float complex z);
23103       long double cimagl(long double complex z);
```

23104 **DESCRIPTION**

23105 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23106 conflict between the requirements described here and the ISO C standard is unintentional. This
 23107 volume of POSIX.1-2017 defers to the ISO C standard.

23108 These functions shall compute the imaginary part of *z*.23109 **RETURN VALUE**

23110 These functions shall return the imaginary part value (as a real).

23111 **ERRORS**

23112 No errors are defined.

23113 **EXAMPLES**

23114 None.

23115 **APPLICATION USAGE**23116 For a variable *z* of complex type:23117 `z == creal(z) + cimag(z)*I`23118 **RATIONALE**

23119 None.

23120 **FUTURE DIRECTIONS**

23121 None.

23122 **SEE ALSO**23123 *carg()*, *conj()*, *cproj()*, *creal()*23124 XBD [<complex.h>](#)23125 **CHANGE HISTORY**

23126 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23127 **NAME**

23128 clearerr — clear indicators on a stream

23129 **SYNOPSIS**

23130 #include <stdio.h>

23131 void clearerr(FILE *stream);

23132 **DESCRIPTION**

23133 CX The functionality described on this reference page is aligned with the ISO C standard. Any
23134 conflict between the requirements described here and the ISO C standard is unintentional. This
23135 volume of POSIX.1-2017 defers to the ISO C standard.

23136 The `clearerr()` function shall clear the end-of-file and error indicators for the stream to which
23137 `stream` points.

23138 CX The `clearerr()` function shall not change the setting of `errno` if `stream` is valid.

23139 **RETURN VALUE**23140 The `clearerr()` function shall not return a value.23141 **ERRORS**

23142 No errors are defined.

23143 **EXAMPLES**

23144 None.

23145 **APPLICATION USAGE**

23146 None.

23147 **RATIONALE**

23148 None.

23149 **FUTURE DIRECTIONS**

23150 None.

23151 **SEE ALSO**

23152 XBD <stdio.h>

23153 **CHANGE HISTORY**

23154 First released in Issue 1. Derived from Issue 1 of the SVID.

23155 **Issue 7**

23156 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0057 [401] is applied.

23157 **NAME**

23158 clock — report CPU time used

23159 **SYNOPSIS**

23160 #include <time.h>

23161 clock_t clock(void);

23162 **DESCRIPTION**

23163 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23164 conflict between the requirements described here and the ISO C standard is unintentional. This
 23165 volume of POSIX.1-2017 defers to the ISO C standard.

23166 The *clock()* function shall return the implementation's best approximation to the processor time
 23167 used by the process since the beginning of an implementation-defined era related only to the
 23168 process invocation.

23169 **RETURN VALUE**

23170 To determine the time in seconds, the value returned by *clock()* should be divided by the value
 23171 XSI of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.
 23172 If the processor time used is not available or its value cannot be represented, the function shall
 23173 return the value `(clock_t)-1`.

23174 **ERRORS**

23175 No errors are defined.

23176 **EXAMPLES**

23177 None.

23178 **APPLICATION USAGE**

23179 In programming environments where `clock_t` is a 32-bit integer type and `CLOCKS_PER_SEC` is
 23180 one million, *clock()* will start failing in less than 36 minutes of processor time for signed `clock_t`,
 23181 or 72 minutes for unsigned `clock_t`. Applications intended to be portable to such environments
 23182 should use *times()* instead (or *clock_gettime()* with `CLOCK_PROCESS_CPUTIME_ID`, if
 23183 supported).

23184 In order to measure the time spent in a program, *clock()* should be called at the start of the
 23185 program and its return value subtracted from the value returned by subsequent calls. The value
 23186 returned by *clock()* is defined for compatibility across systems that have clocks with different
 23187 resolutions. The resolution on any particular system need not be to microsecond accuracy.

23188 **RATIONALE**

23189 None.

23190 **FUTURE DIRECTIONS**

23191 None.

23192 **SEE ALSO**

23193 *asctime()*, *clock_getres()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*,
 23194 *time()*, *times()*, *utime()*

23195 XBD `<time.h>`23196 **CHANGE HISTORY**

23197 First released in Issue 1. Derived from Issue 1 of the SVID.

23198 **Issue 7**
23199

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0067 [686] is applied.

23200 **NAME**23201 clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)23202 **SYNOPSIS**

```
23203 CPT #include <time.h>
23204 int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

23205 **DESCRIPTION**

23206 The *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process
 23207 specified by *pid*. If the process described by *pid* exists and the calling process has permission, the
 23208 clock ID of this clock shall be returned in *clock_id*.

23209 If *pid* is zero, the *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of
 23210 the process making the call, in *clock_id*.

23211 The conditions under which one process has permission to obtain the CPU-time clock ID of
 23212 other processes are implementation-defined.

23213 **RETURN VALUE**

23214 Upon successful completion, *clock_getcpuclockid()* shall return zero; otherwise, an error number
 23215 shall be returned to indicate the error.

23216 **ERRORS**

23217 The *clock_getcpuclockid()* function shall fail if:

23218 [EPERM] The requesting process does not have permission to access the CPU-time clock
 23219 for the process.

23220 The *clock_getcpuclockid()* function may fail if:

23221 [ESRCH] No process can be found corresponding to the process specified by *pid*.

23222 **EXAMPLES**

23223 None.

23224 **APPLICATION USAGE**

23225 The *clock_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not be
 23226 provided on all implementations.

23227 **RATIONALE**

23228 None.

23229 **FUTURE DIRECTIONS**

23230 None.

23231 **SEE ALSO**

23232 [clock_getres\(\)](#), [timer_create\(\)](#)

23233 XBD [<time.h>](#)

23234 **CHANGE HISTORY**

23235 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

23236 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

23237 **NAME**

23238 clock_getres, clock_gettime, clock_settime — clock and timer functions

23239 **SYNOPSIS**

```

23240 CX #include <time.h>
23241 int clock_getres(clockid_t clock_id, struct timespec *res);
23242 int clock_gettime(clockid_t clock_id, struct timespec *tp);
23243 int clock_settime(clockid_t clock_id, const struct timespec *tp);

```

23244 **DESCRIPTION**

23245 The `clock_getres()` function shall return the resolution of any clock. Clock resolutions are
 23246 implementation-defined and cannot be set by a process. If the argument `res` is not NULL, the
 23247 resolution of the specified clock shall be stored in the location pointed to by `res`. If `res` is NULL,
 23248 the clock resolution is not returned. If the `time` argument of `clock_settime()` is not a multiple of `res`,
 23249 then the value is truncated to a multiple of `res`.

23250 The `clock_gettime()` function shall return the current value `tp` for the specified clock, `clock_id`.

23251 The `clock_settime()` function shall set the specified clock, `clock_id`, to the value specified by `tp`.
 23252 Time values that are between two consecutive non-negative integer multiples of the resolution of
 23253 the specified clock shall be truncated down to the smaller multiple of the resolution.

23254 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that
 23255 is meaningful only within a process). All implementations shall support a `clock_id` of
 23256 CLOCK_REALTIME as defined in `<time.h>`. This clock represents the clock measuring real time
 23257 for the system. For this clock, the values returned by `clock_gettime()` and specified by
 23258 `clock_settime()` represent the amount of time (in seconds and nanoseconds) since the Epoch. An
 23259 implementation may also support additional clocks. The interpretation of time values for these
 23260 clocks is unspecified.

23261 If the value of the CLOCK_REALTIME clock is set via `clock_settime()`, the new value of the clock
 23262 shall be used to determine the time of expiration for absolute time services based upon the
 23263 CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the
 23264 absolute time requested at the invocation of such a time service is before the new value of the
 23265 clock, the time service shall expire immediately as if the clock had reached the requested time
 23266 normally.

23267 Setting the value of the CLOCK_REALTIME clock via `clock_settime()` shall have no effect on
 23268 threads that are blocked waiting for a relative time service based upon this clock, including the
 23269 `nanosleep()` function; nor on the expiration of relative timers based upon this clock.
 23270 Consequently, these time services shall expire when the requested relative interval elapses,
 23271 independently of the new or old value of the clock.

23272 MON If the Monotonic Clock option is supported, all implementations shall support a `clock_id` of
 23273 CLOCK_MONOTONIC defined in `<time.h>`. This clock represents the monotonic clock for the
 23274 system. For this clock, the value returned by `clock_gettime()` represents the amount of time (in
 23275 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up
 23276 time, or the Epoch). This point does not change after system start-up time. The value of the
 23277 CLOCK_MONOTONIC clock cannot be set via `clock_settime()`. This function shall fail if it is
 23278 invoked with a `clock_id` argument of CLOCK_MONOTONIC.

23279 The effect of setting a clock via `clock_settime()` on armed per-process timers associated with a
 23280 clock other than CLOCK_REALTIME is implementation-defined.

23281 If the value of the CLOCK_REALTIME clock is set via `clock_settime()`, the new value of the clock
 23282 shall be used to determine the time at which the system shall awaken a thread blocked on an

23283 absolute *clock_nanosleep()* call based upon the CLOCK_REALTIME clock. If the absolute time
 23284 requested at the invocation of such a time service is before the new value of the clock, the call
 23285 shall return immediately as if the clock had reached the requested time normally.

23286 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on any
 23287 thread that is blocked on a relative *clock_nanosleep()* call. Consequently, the call shall return
 23288 when the requested relative interval elapses, independently of the new or old value of the clock.

23289 Appropriate privileges to set a particular clock are implementation-defined.

23290 CPT If _POSIX_CPUTIME is defined, implementations shall support clock ID values obtained by
 23291 invoking *clock_getcpuclockid()*, which represent the CPU-time clock of a given process.
 23292 Implementations shall also support the special **clockid_t** value
 23293 CLOCK_PROCESS_CPUTIME_ID, which represents the CPU-time clock of the calling process
 23294 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 23295 returned by *clock_gettime()* and specified by *clock_settime()* represent the amount of execution
 23296 time of the process associated with the clock. Changing the value of a CPU-time clock via
 23297 *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see
 23298 [Scheduling Policies](#)).

23299 TCT If _POSIX_THREAD_CPUTIME is defined, implementations shall support clock ID values
 23300 obtained by invoking *pthread_getcpuclockid()*, which represent the CPU-time clock of a given
 23301 thread. Implementations shall also support the special **clockid_t** value
 23302 CLOCK_THREAD_CPUTIME_ID, which represents the CPU-time clock of the calling thread
 23303 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 23304 returned by *clock_gettime()* and specified by *clock_settime()* shall represent the amount of
 23305 execution time of the thread associated with the clock. Changing the value of a CPU-time clock
 23306 via *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy
 23307 (see [Scheduling Policies](#)).

23308 RETURN VALUE

23309 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that
 23310 an error occurred, and *errno* shall be set to indicate the error.

23311 ERRORS

23312 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions shall fail if:

23313 [EINVAL] The *clock_id* argument does not specify a known clock.

23314 The *clock_gettime()* function shall fail if:

23315 [EOVERFLOW] The number of seconds will not fit in an object of type **time_t**.

23316 The *clock_settime()* function shall fail if:

23317 [EINVAL] The *tp* argument to *clock_settime()* is outside the range for the given clock ID.

23318 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or
 23319 equal to 1 000 million.

23320 MON [EINVAL] The value of the *clock_id* argument is CLOCK_MONOTONIC.

23321 The *clock_settime()* function may fail if:

23322 [EPERM] The requesting process does not have appropriate privileges to set the
 23323 specified clock.

23324 **EXAMPLES**

23325 None.

23326 **APPLICATION USAGE**

23327 Note that the absolute value of the monotonic clock is meaningless (because its origin is
 23328 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the
 23329 fact that the value of this clock is never set and, therefore, that time intervals measured with this
 23330 clock will not be affected by calls to *clock_settime()*.

23331 **RATIONALE**

23332 None.

23333 **FUTURE DIRECTIONS**

23334 None.

23335 **SEE ALSO**

23336 [Scheduling Policies](#) (on page 506), *clock_getcpuclockid()*, *clock_nanosleep()*, *ctime()*, *mq_receive()*,
 23337 *mq_send()*, *nanosleep()*, *pthread_mutex_timedlock()*, *sem_timedwait()*, *time()*, *timer_create()*,
 23338 *timer_getoverrun()*

23339 XBD [<time.h>](#)23340 **CHANGE HISTORY**

23341 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

23342 **Issue 6**

23343 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 23344 implementation does not support the Timers option.

23345 The APPLICATION USAGE section is added.

23346 The following changes were made to align with the IEEE P1003.1a draft standard:

23347 Clarification is added of the effect of resetting the clock resolution.

23348 CPU-time clocks and the *clock_getcpuclockid()* function are added for alignment with IEEE Std
 23349 1003.1d-1999.

23350 The following changes are added for alignment with IEEE Std 1003.1j-2000:

23351 The DESCRIPTION is updated as follows:

23352 ‡ The value returned by *clock_gettime()* for CLOCK_MONOTONIC is specified.23353 ‡ The *clock_settime()* function failing for CLOCK_MONOTONIC is specified.

23354 ‡ The effects of *clock_settime()* on the *clock_nanosleep()* function with respect to
 23355 CLOCK_REALTIME are specified.

23356 An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime()* fails for
 23357 CLOCK_MONOTONIC.

23358 The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not
 23359 and shall not be set by *clock_settime()* since the absolute value of the
 23360 CLOCK_MONOTONIC clock is meaningless.

23361 The *clock_nanosleep()*, *mq_timedreceive()*, *mq_timedsend()*, *pthread_mutex_timedlock()*,
 23362 *sem_timedwait()*, *timer_create()*, and *timer_settime()* functions are added to the SEE ALSO
 23363 section.

23364 **Issue 7**

23365 Functionality relating to the Clock Selection option is moved to the Base.

23366 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions are moved from the Timers
23367 option to the Base.

23368 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0058 [106] is applied.

23369 **NAME**

23370 clock_nanosleep — high resolution sleep with specifiable clock

23371 **SYNOPSIS**

```
23372 CX #include <time.h>
23373
23373 int clock_nanosleep(clockid_t clock_id, int flags,
23374 const struct timespec *rqtp, struct timespec *rmtp);
```

23375 **DESCRIPTION**

23376 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall
 23377 cause the current thread to be suspended from execution until either the time interval specified
 23378 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to
 23379 invoke a signal-catching function, or the process is terminated. The clock used to measure the
 23380 time shall be the clock specified by *clock_id*.

23381 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall
 23382 cause the current thread to be suspended from execution until either the time value of the clock
 23383 specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is
 23384 delivered to the calling thread and its action is to invoke a signal-catching function, or the
 23385 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or
 23386 equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately
 23387 and the calling process shall not be suspended.

23388 The suspension time caused by this function may be longer than requested because the
 23389 argument value is rounded up to an integer multiple of the sleep resolution, or because of the
 23390 scheduling of other activity by the system. But, except for the case of being interrupted by a
 23391 signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the
 23392 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as
 23393 measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function
 23394 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the
 23395 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being
 23396 interrupted by a signal.

23397 The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any
 23398 signal.

23399 The `clock_nanosleep()` function shall fail if the *clock_id* argument refers to the CPU-time clock of
 23400 the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are allowed.

23401 **RETURN VALUE**

23402 If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value
 23403 shall be zero.

23404 If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall
 23405 return the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp*
 23406 argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the
 23407 amount of time remaining in the interval (the requested time minus the time actually slept). The
 23408 *rqtp* and *rmtp* arguments can point to the same object. If the *rmtp* argument is NULL, the
 23409 remaining time is not returned. The absolute `clock_nanosleep()` function has no effect on the
 23410 structure referenced by *rmtp*.

23411 If `clock_nanosleep()` fails, it shall return the corresponding error value.

23412 **ERRORS**23413 The *clock_nanosleep()* function shall fail if:23414 [EINTR] The *clock_nanosleep()* function was interrupted by a signal.23415 [EINVAL] The *rqt*p argument specified a nanosecond value less than zero or greater than
23416 or equal to 1 000 million; or the `TIMER_ABSTIME` flag was specified in *flags*
23417 and the *rqt*p argument is outside the range for the clock specified by *clock_id*;
23418 or the *clock_id* argument does not specify a known clock, or specifies the CPU-
23419 time clock of the calling thread.23420 [ENOTSUP] The *clock_id* argument specifies a clock for which *clock_nanosleep()* is not
23421 supported, such as a CPU-time clock.23422 **EXAMPLES**

23423 None.

23424 **APPLICATION USAGE**23425 Calling *clock_nanosleep()* with the value `TIMER_ABSTIME` not set in the *flags* argument and with
23426 a *clock_id* of `CLOCK_REALTIME` is equivalent to calling *nanosleep()* with the same *rqt*p and *rmt*p
23427 arguments.23428 **RATIONALE**23429 The *nanosleep()* function specifies that the system-wide clock `CLOCK_REALTIME` is used to
23430 measure the elapsed time for this time service. However, with the introduction of the monotonic
23431 clock `CLOCK_MONOTONIC` a new relative sleep function is needed to allow an application to
23432 take advantage of the special characteristics of this clock.23433 There are many applications in which a process needs to be suspended and then activated
23434 multiple times in a periodic way; for example, to poll the status of a non-interrupting device or
23435 to refresh a display device. For these cases, it is known that precise periodic activation cannot be
23436 achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic
23437 process that is activated at time T_0 , executes for a while, and then wants to suspend itself until
23438 time T_0+T , the period being T . If this process wants to use the *nanosleep()* function, it must first
23439 call *clock_gettime()* to get the current time, then calculate the difference between the current time
23440 and T_0+T and, finally, call *nanosleep()* using the computed interval. However, the process could
23441 be preempted by a different process between the two function calls, and in this case the interval
23442 computed would be wrong; the process would wake up later than desired. This problem would
23443 not occur with the absolute *clock_nanosleep()* function, since only one function call would be
23444 necessary to suspend the process until the desired time. In other cases, however, a relative sleep
23445 is needed, and that is why both functionalities are required.23446 Although it is possible to implement periodic processes using the timers interface, this
23447 implementation would require the use of signals, and the reservation of some signal numbers. In
23448 this regard, the reasons for including an absolute version of the *clock_nanosleep()* function in
23449 POSIX.1-2017 are the same as for the inclusion of the relative *nanosleep()*.23450 It is also possible to implement precise periodic processes using *pthread_cond_timedwait()*, in
23451 which an absolute timeout is specified that takes effect if the condition variable involved is never
23452 signaled. However, the use of this interface is unnatural, and involves performing other
23453 operations on mutexes and condition variables that imply an unnecessary overhead.
23454 Furthermore, *pthread_cond_timedwait()* is not available in implementations that do not support
23455 threads.23456 Although the interface of the relative and absolute versions of the new high resolution sleep
23457 service is the same *clock_nanosleep()* function, the *rmt*p argument is only used in the relative
23458 sleep. This argument is needed in the relative *clock_nanosleep()* function to reissue the function

23459 call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep()* function
23460 call; if the call is interrupted by a signal, the absolute *clock_nanosleep()* function can be invoked
23461 again with the same *rqtp* argument used in the interrupted call.

23462 **FUTURE DIRECTIONS**

23463 None.

23464 **SEE ALSO**

23465 *clock_getres()*, *nanosleep()*, *pthread_cond_timedwait()*, *sleep()*

23466 XBD <[time.h](#)>

23467 **CHANGE HISTORY**

23468 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

23469 **Issue 7**

23470 The *clock_nanosleep()* function is moved from the Clock Selection option to the Base.

23471 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0068 [909] is applied.

23472 **NAME**

23473 clock_settime ‡clock and timer functions

23474 **SYNOPSIS**

```
23475 CX #include <time.h>  
23476 int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

23477 **DESCRIPTION**

23478 Refer to [clock_getres\(\)](#).

23479 **NAME**23480 `clog, clogf, clogl` ‡complex natural logarithm functions23481 **SYNOPSIS**23482 `#include <complex.h>`23483 `double complex clog(double complex z);`23484 `float complex clogf(float complex z);`23485 `long double complex clogl(long double complex z);`23486 **DESCRIPTION**

23487 CX The functionality described on this reference page is aligned with the ISO C standard. Any
23488 conflict between the requirements described here and the ISO C standard is unintentional. This
23489 volume of POSIX.1-2017 defers to the ISO C standard.

23490 These functions shall compute the complex natural (base e) logarithm of z , with a branch cut
23491 along the negative real axis.

23492 **RETURN VALUE**

23493 These functions shall return the complex natural logarithm value, in the range of a strip
23494 mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary
23495 axis.

23496 **ERRORS**

23497 No errors are defined.

23498 **EXAMPLES**

23499 None.

23500 **APPLICATION USAGE**

23501 None.

23502 **RATIONALE**

23503 None.

23504 **FUTURE DIRECTIONS**

23505 None.

23506 **SEE ALSO**23507 [cexp\(\)](#)23508 XBD [<complex.h>](#)23509 **CHANGE HISTORY**

23510 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23511 **NAME**

23512 close ‡'close a file descriptor

23513 **SYNOPSIS**

23514 #include <unistd.h>

23515 int close(int *fildev*);23516 **DESCRIPTION**

23517 The *close()* function shall deallocate the file descriptor indicated by *fildev*. To deallocate means to
 23518 make the file descriptor available for return by subsequent calls to *open()* or other functions that
 23519 allocate file descriptors. All outstanding record locks owned by the process on the file associated
 23520 with the file descriptor shall be removed (that is, unlocked).

23521 If *close()* is interrupted by a signal that is to be caught, it shall return -1 with *errno* set to [EINTR]
 23522 and the state of *fildev* is unspecified. If an I/O error occurred while reading from or writing to
 23523 the file system during *close()*, it may return -1 with *errno* set to [EIO]; if this error is returned, the
 23524 state of *fildev* is unspecified.

23525 When all file descriptors associated with a pipe or FIFO special file are closed, any data
 23526 remaining in the pipe or FIFO shall be discarded.

23527 When all file descriptors associated with an open file description have been closed, the open file
 23528 description shall be freed.

23529 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
 23530 space occupied by the file shall be freed and the file shall no longer be accessible.

23531 OB XSR If a STREAMS-based *fildev* is closed and the calling process was previously registered to receive
 23532 a SIGPOLL signal for events associated with that STREAM, the calling process shall be
 23533 unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause
 23534 the STREAM associated with *fildev* to be dismantled. If O_NONBLOCK is not set and there have
 23535 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*
 23536 shall wait for an unspecified time (for each module and driver) for any output to drain before
 23537 dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl()* request. If
 23538 the O_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for
 23539 output to drain, and shall dismantle the STREAM immediately.

23540 If the implementation supports STREAMS-based pipes, and *fildev* is associated with one end of a
 23541 pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the
 23542 other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end
 23543 to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and
 23544 gets detached, the STREAM associated with that end shall also be dismantled.

23545 XSI If *fildev* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal
 23546 shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is
 23547 the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal
 23548 flushes all queued input and output.

23549 OB XSR If *fildev* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
 23550 may be sent to the master.

23551 When there is an outstanding cancelable asynchronous I/O operation against *fildev* when *close()*
 23552 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
 23553 as if the *close()* operation had not yet occurred. All operations that are not canceled shall
 23554 complete as if the *close()* blocked until the operations completed. The *close()* operation itself
 23555 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
 23556 which I/O operation may be canceled upon *close()*, is implementation-defined.

23557 SHM If a memory mapped file or a shared memory object remains referenced at the last close (that is,
 23558 a process has it mapped), then the entire contents of the memory object shall persist until the
 23559 SHM memory object becomes unreferenced. If this is the last close of a memory mapped file or a
 23560 shared memory object and the close results in the memory object becoming unreferenced, and
 23561 the memory object has been unlinked, then the memory object shall be removed.

23562 If *fdes* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in
 23563 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,
 23564 and the socket has untransmitted data, then *close()* shall block for up to the current linger
 23565 interval until all data is transmitted.

23566 RETURN VALUE

23567 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 23568 indicate the error.

23569 ERRORS

23570 The *close()* function shall fail if:

23571 [EBADF] The *fdes* argument is not a open file descriptor.

23572 [EINTR] The *close()* function was interrupted by a signal.

23573 The *close()* function may fail if:

23574 [EIO] An I/O error occurred while reading from or writing to the file system.

23575 EXAMPLES

23576 Reassigning a File Descriptor

23577 The following example closes the file descriptor associated with standard output for the current
 23578 process, re-assigns standard output to a new file descriptor, and closes the original file descriptor
 23579 to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard
 23580 input) is not closed.

```
23581 #include <unistd.h>
23582 ...
23583 int pfd;
23584 ...
23585 close(1);
23586 dup(pfd);
23587 close(pfd);
23588 ...
```

23589 Incidentally, this is exactly what could be achieved using:

```
23590 dup2(pfd, 1);
23591 close(pfd);
```


23592 **Closing a File Descriptor**

23593 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is
 23594 made to associate that file descriptor with a stream.

```

23595 #include <stdio.h>
23596 #include <unistd.h>
23597 #include <stdlib.h>

23598 #define LOCKFILE "/etc/ptmp"
23599 ...
23600 int pfd;
23601 FILE *fpfd;
23602 ...
23603 if ((fpfd = fdopen (pfd, "w")) == NULL) {
23604     close(pfd);
23605     unlink(LOCKFILE);
23606     exit(1);
23607 }
23608 ...

```

23609 **APPLICATION USAGE**

23610 An application that had used the *stdio* routine *fopen()* to open a file should use the
 23611 corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no
 23612 longer exists, since the integer corresponding to it no longer refers to a file.

23613 Implementations may use file descriptors that must be inherited into child processes for the
 23614 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,
 23615 an application that calls *close()* on an arbitrary integer risks non-conforming behavior, and
 23616 *close()* can only portably be used on file descriptor values that the application has obtained
 23617 through explicit actions, as well as the three file descriptors corresponding to the standard file
 23618 streams. In multi-threaded parent applications, the practice of calling *close()* in a loop after *fork()*
 23619 and before an *exec* call in order to avoid a race condition of leaking an unintended file descriptor
 23620 into a child process, is therefore unsafe, and the race should instead be combatted by opening all
 23621 file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be
 23622 inherited across *exec*.

23623 Usage of *close()* on file descriptors STDIN_FILENO, STDOUT_FILENO, or STDERR_FILENO
 23624 should immediately be followed by an operation to reopen these file descriptors. Unexpected
 23625 behavior will result if any of these file descriptors is left in a closed state (for example, an
 23626 [EBADF] error from *perror()*) or if an unrelated *open()* or similar call later in the application
 23627 accidentally allocates a file to one of these well-known file descriptors. Furthermore, a *close()*
 23628 followed by a reopen operation (e.g., *open()*, *dup()*, etc.) is not atomic; *dup2()* should be used to
 23629 change standard file descriptors.

23630 **RATIONALE**

23631 The use of interruptible device close routines should be discouraged to avoid problems with the
 23632 implicit closes of file descriptors by *exec* and *exit()*. This volume of POSIX.1-2017 only intends to
 23633 permit such behavior by specifying the [EINTR] error condition.

23634 Note that the requirement for *close()* on a socket to block for up to the current linger interval is
 23635 not conditional on the O_NONBLOCK setting.

23636 The standard developers rejected a proposal to add *closefrom()* to the standard. Because the
 23637 standard permits implementations to use inherited file descriptors as a means of providing a
 23638 conforming environment for the child process, it is not possible to standardize an interface that

23639 closes arbitrary file descriptors above a certain value while still guaranteeing a conforming
23640 environment.

23641 **FUTURE DIRECTIONS**

23642 None.

23643 **SEE ALSO**

23644 [Section 2.6](#) (on page 500), [dup\(\)](#), [exec](#), [exit\(\)](#), [fattach\(\)](#), [fclose\(\)](#), [fdetach\(\)](#), [fopen\(\)](#), [fork\(\)](#), [ioctl\(\)](#),
23645 [open\(\)](#), [perror\(\)](#), [unlink\(\)](#)

23646 XBD [<unistd.h>](#)

23647 **CHANGE HISTORY**

23648 First released in Issue 1. Derived from Issue 1 of the SVID.

23649 **Issue 5**

23650 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

23651 **Issue 6**

23652 The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of
23653 the XSI STREAMS Option Group.

23654 The following new requirements on POSIX implementations derive from alignment with the
23655 Single UNIX Specification:

23656 The [EIO] error condition is added as an optional error.

23657 The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as
23658 unspecified if an I/O error occurs and an [EIO] error condition is returned.

23659 Text referring to sockets is added to the DESCRIPTION.

23660 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
23661 shared memory objects and memory mapped files (and not typed memory objects) are the types
23662 of memory objects to which the paragraph on last closes applies.

23663 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded
23664 text relating to the master side of a pseudo-terminal. The reason for the change is that the
23665 behavior of pseudo-terminals and regular terminals should be as much alike as possible in this
23666 case; the change achieves that and matches historical behavior.

23667 **Issue 7**

23668 Functionality relating to the XSI STREAMS option is marked obsolescent.

23669 Functionality relating to the Asynchronous Input and Output and Memory Mapped Files
23670 options is moved to the Base.

23671 Austin Group Interpretation 1003.1-2001 #139 is applied, clarifying that the requirement for
23672 *close()* on a socket to block for up to the current linger interval is not conditional on the
23673 O_NONBLOCK setting.

23674 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0059 [419], XSH/TC1-2008/0060 [149],
23675 and XSH/TC1-2008/0061 [149] are applied.

23676 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0069 [555] is applied.

23677 **NAME**

23678 closedir — close a directory stream

23679 **SYNOPSIS**

23680 #include <dirent.h>

23681 int closedir(DIR *dirp);

23682 **DESCRIPTION**

23683 The *closedir()* function shall close the directory stream referred to by the argument *dirp*. Upon
 23684 return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file
 23685 descriptor is used to implement type **DIR**, that file descriptor shall be closed.

23686 **RETURN VALUE**

23687 Upon successful completion, *closedir()* shall return 0; otherwise, -1 shall be returned and *errno*
 23688 set to indicate the error.

23689 **ERRORS**23690 The *closedir()* function may fail if:23691 [EBADF] The *dirp* argument does not refer to an open directory stream.23692 [EINTR] The *closedir()* function was interrupted by a signal.23693 **EXAMPLES**23694 **Closing a Directory Stream**23695 The following program fragment demonstrates how the *closedir()* function is used.

```

23696           ...
23697           DIR *dir;
23698           struct dirent *dp;
23699           ...
23700           if ((dir = opendir(".")) == NULL) {
23701           ...
23702           }
23703           while ((dp = readdir(dir)) != NULL) {
23704           ...
23705           }
23706           closedir(dir);
23707           ...

```

23708 **APPLICATION USAGE**

23709 None.

23710 **RATIONALE**

23711 None.

23712 **FUTURE DIRECTIONS**

23713 None.

23714 **SEE ALSO**23715 *dirfd()*, *fdopendir()*

23716 XBD <dirent.h>

23717 **CHANGE HISTORY**

23718 First released in Issue 2.

23719 **Issue 6**23720 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.23721 The following new requirements on POSIX implementations derive from alignment with the
23722 Single UNIX Specification:23723 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
23724 required for conforming implementations of previous POSIX specifications, it was not
23725 required for UNIX applications.

23726 The [EINTR] error condition is added as an optional error condition.

23727 **NAME**

23728 closelog, openlog, setlogmask, syslog — control system log

23729 **SYNOPSIS**

```

23730 XSI #include <syslog.h>
23731 void closelog(void);
23732 void openlog(const char *ident, int logopt, int facility);
23733 int setlogmask(int maskpri);
23734 void syslog(int priority, const char *message, ... /* arguments */);

```

23735 **DESCRIPTION**

23736 The *syslog()* function shall send a message to an implementation-defined logging facility, which
 23737 may log it in an implementation-defined system log, write it to the system console, forward it to
 23738 a list of users, or forward it to the logging facility on another host over the network. The logged
 23739 message shall include a message header and a message body. The message header contains at
 23740 least a timestamp and a tag string.

23741 The message body is generated from the *message* and following arguments in the same manner
 23742 as if these were arguments to *printf()*, except that the additional conversion specification *%m*
 23743 shall be recognized; it shall convert no arguments, shall cause the output of the error message
 23744 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument
 23745 specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion
 23746 specifier character is not just *%m*, the behavior is undefined. A trailing *<newline>* may be added
 23747 if needed.

23748 Values of the *priority* argument are formed by OR'ing together a severity-level value and an
 23749 optional facility value. If no facility value is specified, the current default facility value is used.

23750 Possible values of severity level include:

23751	LOG_EMERG	A panic condition.
23752	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
23753		
23754	LOG_CRIT	Critical conditions, such as hard device errors.
23755	LOG_ERR	Errors.
23756	LOG_WARNING	
23757		Warning messages.
23758	LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
23759		
23760	LOG_INFO	Informational messages.
23761	LOG_DEBUG	Messages that contain information normally of use only when debugging a program.
23762		

23763 The facility indicates the application or system component generating the message. Possible
 23764 facility values include:

23765	LOG_USER	Messages generated by arbitrary processes. This is the default facility identifier if none is specified.
23766		
23767	LOG_LOCAL0	Reserved for local use.

23768	LOG_LOCAL1	Reserved for local use.
23769	LOG_LOCAL2	Reserved for local use.
23770	LOG_LOCAL3	Reserved for local use.
23771	LOG_LOCAL4	Reserved for local use.
23772	LOG_LOCAL5	Reserved for local use.
23773	LOG_LOCAL6	Reserved for local use.
23774	LOG_LOCAL7	Reserved for local use.
23775	The <i>openlog()</i> function shall set process attributes that affect subsequent calls to <i>syslog()</i> . The	
23776	<i>ident</i> argument is a string that is prepended to every message. The <i>logopt</i> argument indicates	
23777	logging options. Values for <i>logopt</i> are constructed by a bitwise-inclusive OR of zero or more of	
23778	the following:	
23779	LOG_PID	Log the process ID with each message. This is useful for identifying specific
23780		processes.
23781	LOG_CONS	Write messages to the system console if they cannot be sent to the logging
23782		facility. The <i>syslog()</i> function ensures that the process does not acquire the
23783		console as a controlling terminal in the process of writing the message.
23784	LOG_NDELAY	Open the connection to the logging facility immediately. Normally the open is
23785		delayed until the first message is logged. This is useful for programs that need
23786		to manage the order in which file descriptors are allocated.
23787	LOG_ODELAY	Delay open until <i>syslog()</i> is called.
23788	LOG_NOWAIT	Do not wait for child processes that may have been created during the course
23789		of logging the message. This option should be used by processes that enable
23790		notification of child termination using SIGCHLD, since <i>syslog()</i> may
23791		otherwise block waiting for a child whose exit status has already been
23792		collected.
23793	The <i>facility</i> argument encodes a default facility to be assigned to all messages that do not have an	
23794	explicit facility already encoded. The initial default facility is LOG_USER.	
23795	The <i>openlog()</i> and <i>syslog()</i> functions may allocate a file descriptor. It is not necessary to call	
23796	<i>openlog()</i> prior to calling <i>syslog()</i> .	
23797	The <i>closelog()</i> function shall close any open file descriptors allocated by previous calls to	
23798	<i>openlog()</i> or <i>syslog()</i> .	
23799	The <i>setlogmask()</i> function shall set the log priority mask for the current process to <i>maskpri</i> and	
23800	return the previous mask. If the <i>maskpri</i> argument is 0, the current log mask is not modified.	
23801	Calls by the current process to <i>syslog()</i> with a priority not set in <i>maskpri</i> shall be rejected. The	
23802	default log mask allows all priorities to be logged. A call to <i>openlog()</i> is not required prior to	
23803	calling <i>setlogmask()</i> .	
23804	Symbolic constants for use as values of the <i>logopt</i> , <i>facility</i> , <i>priority</i> , and <i>maskpri</i> arguments are	
23805	defined in the <syslog.h> header.	
23806	RETURN VALUE	
23807	The <i>setlogmask()</i> function shall return the previous log priority mask. The <i>closelog()</i> , <i>openlog()</i> ,	
23808	and <i>syslog()</i> functions shall not return a value.	

23809 **ERRORS**

23810 None errors are defined.

23811 **EXAMPLES**23812 **Using openlog()**

23813 The following example causes subsequent calls to *syslog()* to log the process ID with each
23814 message, and to write messages to the system console if they cannot be sent to the logging
23815 facility.

```
23816       #include <syslog.h>
23817
23817       char *ident = "Process demo";
23818       int logopt = LOG_PID | LOG_CONS;
23819       int facility = LOG_USER;
23820       ...
23821       openlog(ident, logopt, facility);
```

23822 **Using setlogmask()**

23823 The following example causes subsequent calls to *syslog()* to accept error messages, and to reject
23824 all other messages.

```
23825       #include <syslog.h>
23826
23826       int result;
23827       int mask = LOG_MASK (LOG_ERR);
23828       ...
23829       result = setlogmask(mask);
```

23830 **Using syslog**

23831 The following example sends the message "This is a message" to the default logging
23832 facility, marking the message as an error message generated by random processes.

```
23833       #include <syslog.h>
23834
23834       char *message = "This is a message";
23835       int priority = LOG_ERR | LOG_USER;
23836       ...
23837       syslog(priority, message);
```

23838 **APPLICATION USAGE**

23839 None.

23840 **RATIONALE**

23841 None.

23842 **FUTURE DIRECTIONS**

23843 None.

23844 **SEE ALSO**23845 *fprintf()*23846 XBD *<syslog.h>*

23847 **CHANGE HISTORY**

23848 First released in Issue 4, Version 2.

23849 **Issue 5**

23850 Moved from X/OPEN UNIX extension to BASE.

23851 **Issue 6**

23852 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES
23853 section.

23854 **NAME**

23855 confstr ‡get configurable variables

23856 **SYNOPSIS**

23857 #include <unistd.h>

23858 size_t confstr(int name, char *buf, size_t len);

23859 **DESCRIPTION**23860 The *confstr()* function shall return configuration-defined string values. Its use and purpose are
23861 similar to *sysconf()*, but it is used where string values rather than numeric values are returned.23862 The *name* argument represents the system variable to be queried. The implementation shall
23863 support the following name values, defined in <unistd.h>. It may support others:

23864 _CS_PATH
 23865 _CS_POSIX_V7_ILP32_OFF32_CFLAGS
 23866 _CS_POSIX_V7_ILP32_OFF32_LDFLAGS
 23867 _CS_POSIX_V7_ILP32_OFF32_LIBS
 23868 _CS_POSIX_V7_ILP32_OFFBIG_CFLAGS
 23869 _CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS
 23870 _CS_POSIX_V7_ILP32_OFFBIG_LIBS
 23871 _CS_POSIX_V7_LP64_OFF64_CFLAGS
 23872 _CS_POSIX_V7_LP64_OFF64_LDFLAGS
 23873 _CS_POSIX_V7_LP64_OFF64_LIBS
 23874 _CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS
 23875 _CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS
 23876 _CS_POSIX_V7_LPBIG_OFFBIG_LIBS
 23877 _CS_POSIX_V7_THREADS_CFLAGS
 23878 _CS_POSIX_V7_THREADS_LDFLAGS
 23879 _CS_POSIX_V7_WIDTH_RESTRICTED_ENVS
 23880 _CS_V7_ENV
 23881 OB _CS_POSIX_V6_ILP32_OFF32_CFLAGS
 23882 _CS_POSIX_V6_ILP32_OFF32_LDFLAGS
 23883 _CS_POSIX_V6_ILP32_OFF32_LIBS
 23884 _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
 23885 _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
 23886 _CS_POSIX_V6_ILP32_OFFBIG_LIBS
 23887 _CS_POSIX_V6_LP64_OFF64_CFLAGS
 23888 _CS_POSIX_V6_LP64_OFF64_LDFLAGS
 23889 _CS_POSIX_V6_LP64_OFF64_LIBS
 23890 _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
 23891 _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
 23892 _CS_POSIX_V6_LPBIG_OFFBIG_LIBS
 23893 _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
 23894 _CS_V6_ENV

23895 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into
 23896 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,
 23897 including the terminating null, then *confstr()* shall truncate the string to *len*–1 bytes and null-
 23898 terminate the result. The application can detect that the string was truncated by comparing the
 23899 value returned by *confstr()* with *len*.

23900 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined
 23901 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is

23902 unspecified.

23903 After a call to:

```
23904 confstr(_CS_V7_ENV, buf, sizeof(buf))
```

23905 the string stored in *buf* shall contain a <space>-separated list of the variable=value environment
23906 variable pairs an implementation requires as part of specifying a conforming environment, as
23907 described in the implementations' conformance documentation.

23908 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

```
23909 confstr(_CS_PATH, buf, sizeof(buf))
```

23910 can be used as a value of the *PATH* environment variable that accesses all of the standard
23911 utilities of POSIX.1-2017, that are provided in a manner accessible via the *exec* family of
23912 functions, if the return value is less than or equal to *sizeof(buf)*.

23913 RETURN VALUE

23914 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be
23915 needed to hold the entire configuration-defined value including the terminating null. If this
23916 return value is greater than *len*, the string returned in *buf* is truncated.

23917 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

23918 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*
23919 unchanged.

23920 ERRORS

23921 The *confstr()* function shall fail if:

23922 [EINVAL] The value of the *name* argument is invalid.

23923 EXAMPLES

23924 None.

23925 APPLICATION USAGE

23926 An application can distinguish between an invalid *name* parameter value and one that
23927 corresponds to a configurable variable that has no configuration-defined value by checking if
23928 *errno* is modified. This mirrors the behavior of *sysconf()*.

23929 The original need for this function was to provide a way of finding the configuration-defined
23930 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to
23931 include directories that could contain utilities replacing the standard utilities in the Shell and
23932 Utilities volume of POSIX.1-2017, applications need a way to determine the system-supplied
23933 *PATH* environment variable value that contains the correct search path for the standard utilities.

23934 An application could use:

```
23935 confstr(name, (char *)NULL, (size_t)0)
```

23936 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to
23937 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,
23938 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use
23939 *malloc()* to allocate a larger buffer if it finds that this is too small.

23940 RATIONALE

23941 Application developers can normally determine any configuration variable by means of reading
23942 from the stream opened by a call to:

```
23943 popen("command -p getconf variable", "r");
```

23944 The *confstr()* function with a *name* argument of `_CS_PATH` returns a string that can be used as a
 23945 *PATH* environment variable setting that will reference the standard shell and utilities as
 23946 described in the Shell and Utilities volume of POSIX.1-2017.

23947 The *confstr()* function copies the returned string into a buffer supplied by the application instead
 23948 of returning a pointer to a string. This allows a cleaner function in some implementations (such
 23949 as those with lightweight threads) and resolves questions about when the application must copy
 23950 the string returned.

23951 FUTURE DIRECTIONS

23952 None.

23953 SEE ALSO

23954 *exec*, *fpathconf()*, *sysconf()*

23955 XBD <*unistd.h*>

23956 XCU *c99*

23957 CHANGE HISTORY

23958 First released in Issue 4. Derived from the ISO POSIX-2 standard.

23959 Issue 5

23960 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those
 23961 marked EX are new in this version.

23962 Issue 6

23963 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the
 23964 size of the buffer now explicitly states that this includes the terminating null.

23965 The following new requirements on POSIX implementations derive from alignment with the
 23966 Single UNIX Specification:

23967 The DESCRIPTION is updated with new arguments which can be used to determine
 23968 configuration strings for C compiler flags, linker/loader flags, and libraries for each
 23969 different supported programming environment. This is a change to support data size
 23970 neutrality.

23971 The following changes were made to align with the IEEE P1003.1a draft standard:

23972 The DESCRIPTION is updated to include text describing how `_CS_PATH` can be used to
 23973 obtain a *PATH* to access the standard utilities.

23974 The macros associated with the *c89* programming models are marked LEGACY and new
 23975 equivalent macros associated with *c99* are introduced.

23976 Issue 7

23977 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the `_CS_V7_ENV` variable.

23978 Austin Group Interpretations 1003.1-2001 #166 is applied to permit an additional compiler flag
 23979 to enable threads.

23980 The V6 variables for the supported programming environments are marked obsolescent.

23981 The variables for the supported programming environments are updated to be V7.

23982 The LEGACY variables and obsolescent values are removed.

23983 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0070 [810] and XSH/TC2-2008/0071
 23984 [911] are applied.

23985 **NAME**

23986 conj, conjf, conjl ‡complex conjugate functions

23987 **SYNOPSIS**

23988 #include <complex.h>

23989 double complex conj(double complex z);

23990 float complex conjf(float complex z);

23991 long double complex conjl(long double complex z);

23992 **DESCRIPTION**

23993 CX The functionality described on this reference page is aligned with the ISO C standard. Any
23994 conflict between the requirements described here and the ISO C standard is unintentional. This
23995 volume of POSIX.1-2017 defers to the ISO C standard.

23996 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary
23997 part.

23998 **RETURN VALUE**

23999 These functions return the complex conjugate value.

24000 **ERRORS**

24001 No errors are defined.

24002 **EXAMPLES**

24003 None.

24004 **APPLICATION USAGE**

24005 None.

24006 **RATIONALE**

24007 None.

24008 **FUTURE DIRECTIONS**

24009 None.

24010 **SEE ALSO**24011 [carg\(\)](#), [cimag\(\)](#), [cproj\(\)](#), [creal\(\)](#)24012 XBD [<complex.h>](#)24013 **CHANGE HISTORY**

24014 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24015 **NAME**

24016 connect ‡connect a socket

24017 **SYNOPSIS**

24018 #include <sys/socket.h>

24019 int connect(int *socket*, const struct sockaddr **address*,
24020 socklen_t *address_len*);24021 **DESCRIPTION**24022 The *connect()* function shall attempt to make a connection on a connection-mode socket or to set
24023 or reset the peer address of a connectionless-mode socket. The function takes the following
24024 arguments:

24025	<i>socket</i>	Specifies the file descriptor associated with the socket.
24026	<i>address</i>	Points to a sockaddr structure containing the peer address. The length and format of the address depend on the address family of the socket.
24028	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> argument.

24030 If the socket has not already been bound to a local address, *connect()* shall bind it to an address
24031 which, unless the socket's address family is AF_UNIX, is an unused local address.24032 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,
24033 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
24034 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent
24035 *recv()* functions. If the *sa_family* member of *address* is AF_UNSPEC, the socket's peer address
24036 shall be reset. Note that despite no connection being made, the term "connected" is used to
24037 describe a connectionless-mode socket for which a peer address has been set.24038 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection
24039 to the address specified by the *address* argument. If the connection cannot be established
24040 immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall
24041 block for up to an unspecified timeout interval until the connection is established. If the timeout
24042 interval expires before the connection is established, *connect()* shall fail and the connection
24043 attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked
24044 waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection
24045 request shall not be aborted, and the connection shall be established asynchronously.24046 If the connection cannot be established immediately and O_NONBLOCK is set for the file
24047 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection
24048 request shall not be aborted, and the connection shall be established asynchronously. Subsequent
24049 calls to *connect()* for the same socket, before the connection is established, shall fail and set *errno*
24050 to [EALREADY].24051 When the connection has been established asynchronously, *pselect()*, *select()*, and *poll()* shall
24052 indicate that the file descriptor for the socket is ready for writing.24053 The socket in use may require the process to have appropriate privileges to use the *connect()*
24054 function.24055 **RETURN VALUE**24056 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*
24057 set to indicate the error.

24058 **ERRORS**

- 24059 The *connect()* function shall fail if:
- 24060 [EADDRNOTAVAIL]
24061 The specified address is not available from the local machine.
- 24062 [EAFNOSUPPORT]
24063 The specified address is not a valid address for the address family of the
24064 specified socket.
- 24065 [EALREADY] A connection request is already in progress for the specified socket.
- 24066 [EBADF] The *socket* argument is not a valid file descriptor.
- 24067 [ECONNREFUSED]
24068 The target address was not listening for connections or refused the connection
24069 request.
- 24070 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the connection
24071 cannot be immediately established; the connection shall be established
24072 asynchronously.
- 24073 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal
24074 that was caught; the connection shall be established asynchronously.
- 24075 [EISCONN] The specified socket is connection-mode and is already connected.
- 24076 [ENETUNREACH]
24077 No route to the network is present.
- 24078 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 24079 [EPROTOTYPE] The specified address has a different type than the socket bound to the
24080 specified peer address.
- 24081 [ETIMEDOUT] The attempt to connect timed out before a connection was made.
- 24082 If the address family of the socket is AF_UNIX, then *connect()* shall fail if:
- 24083 [EIO] An I/O error occurred while reading from or writing to the file system.
- 24084 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
24085 in *address*.
- 24086 [ENAMETOOLONG]
24087 The length of a component of a pathname is longer than {NAME_MAX}.
- 24088 [ENOENT] A component of the pathname does not name an existing file or the pathname
24089 is an empty string.
- 24090 [ENOTDIR] A component of the path prefix of the pathname in *address* names an existing
24091 file that is neither a directory nor a symbolic link to a directory, or the
24092 pathname in *address* contains at least one non-*</i>slash*>* character and ends with
24093 one or more trailing *</i>slash*>* characters and the last pathname component
24094 names an existing file that is neither a directory nor a symbolic link to a
24095 directory.**
- 24096 The *connect()* function may fail if:
- 24097 [EACCES] Search permission is denied for a component of the path prefix; or write access
24098 to the named socket is denied.

- 24099 [EADDRINUSE] Attempt to establish a connection that uses addresses that are already in use.
- 24100 [ECONNRESET] Remote host reset the connection request.
- 24101 [EHOSTUNREACH]
- 24102 The destination host cannot be reached (probably because the host is down or
- 24103 a remote router cannot reach it).
- 24104 [EINVAL] The *address_len* argument is not a valid length for the address family; or
- 24105 invalid address family in the **sockaddr** structure.
- 24106 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
- 24107 resolution of the pathname in *address*.
- 24108 [ENAMETOOLONG]
- 24109 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
- 24110 symbolic link produced an intermediate result with a length that exceeds
- 24111 {PATH_MAX}.
- 24112 [ENETDOWN] The local network interface used to reach the destination is down.
- 24113 [ENOBUFS] No buffer space is available.
- 24114 [EOPNOTSUPP] The socket is listening and cannot be connected.

24115 EXAMPLES

24116 None.

24117 APPLICATION USAGE

24118 If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the

24119 file descriptor and create a new socket before attempting to reconnect.

24120 RATIONALE

24121 None.

24122 FUTURE DIRECTIONS

24123 None.

24124 SEE ALSO

24125 *accept()*, *bind()*, *close()*, *getsockname()*, *poll()*, *pselect()*, *send()*, *shutdown()*, *socket()*

24126 XBD <[sys/socket.h](#)>

24127 CHANGE HISTORY

24128 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

24129 The wording of the mandatory [ELOOP] error condition is updated, and a second optional

24130 [ELOOP] error condition is added.

24131 Issue 7

24132 Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected

24133 sockets.

24134 Austin Group Interpretation 1003.1-2001 #143 is applied.

24135 Austin Group Interpretation 1003.1-2001 #188 is applied, changing the method used to reset a

24136 peer address for a datagram socket.

24137 SD5-XSH-ERN-185 is applied.

24138 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0062 [324] is applied.

24139 **NAME**

24140 copysign, copysignf, copysignl ‡number manipulation function

24141 **SYNOPSIS**

24142 #include <math.h>

24143 double copysign(double *x*, double *y*);24144 float copysignf(float *x*, float *y*);24145 long double copysignl(long double *x*, long double *y*);24146 **DESCRIPTION**

24147 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24148 conflict between the requirements described here and the ISO C standard is unintentional. This
24149 volume of POSIX.1-2017 defers to the ISO C standard.

24150 These functions shall produce a value with the magnitude of *x* and the sign of *y*. On
24151 implementations that represent a signed zero but do not treat negative zero consistently in
24152 arithmetic operations, these functions regard the sign of zero as positive.

24153 **RETURN VALUE**

24154 Upon successful completion, these functions shall return a value with the magnitude of *x* and
24155 the sign of *y*.

24156 **ERRORS**

24157 No errors are defined.

24158 **EXAMPLES**

24159 None.

24160 **APPLICATION USAGE**

24161 None.

24162 **RATIONALE**

24163 None.

24164 **FUTURE DIRECTIONS**

24165 None.

24166 **SEE ALSO**24167 [signbit\(\)](#)24168 XBD [<math.h>](#)24169 **CHANGE HISTORY**

24170 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24171 **NAME**

24172 cos, cosf, cosl ‡cosine function

24173 **SYNOPSIS**

```
24174 #include <math.h>
24175 double cos(double x);
24176 float cosf(float x);
24177 long double cosl(long double x);
```

24178 **DESCRIPTION**

24179 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24180 conflict between the requirements described here and the ISO C standard is unintentional. This
 24181 volume of POSIX.1-2017 defers to the ISO C standard.

24182 These functions shall compute the cosine of their argument x , measured in radians.

24183 An application wishing to check for error situations should set *errno* to zero and call
 24184 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 24185 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 24186 zero, an error has occurred.

24187 **RETURN VALUE**

24188 Upon successful completion, these functions shall return the cosine of x .

24189 MX If x is NaN, a NaN shall be returned.

24190 If x is ± 0 , the value 1.0 shall be returned.

24191 If x is $\pm\text{Inf}$, a domain error shall occur, and a NaN shall be returned.

24192 **ERRORS**

24193 These functions shall fail if:

24194 MX **Domain Error** The x argument is $\pm\text{Inf}$.

24195 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 24196 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 24197 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 24198 shall be raised.

24199 **EXAMPLES**24200 **Taking the Cosine of a 45-Degree Angle**

```
24201 #include <math.h>
24202 ...
24203 double radians = 45 * M_PI / 180;
24204 double result;
24205 ...
24206 result = cos(radians);
```

24207 **APPLICATION USAGE**

24208 These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far
 24209 from 0.

24210 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 24211 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

24212 **RATIONALE**

24213 None.

24214 **FUTURE DIRECTIONS**

24215 None.

24216 **SEE ALSO**24217 *acos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, *tan()*24218 XBD Section 4.20 (on page 117), **<math.h>**24219 **CHANGE HISTORY**

24220 First released in Issue 1. Derived from Issue 1 of the SVID.

24221 **Issue 5**24222 The DESCRIPTION is updated to indicate how an application should check for an error. This
24223 text was previously published in the APPLICATION USAGE section.24224 **Issue 6**24225 The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.24226 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
24227 revised to align with the ISO/IEC 9899:1999 standard.24228 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
24229 marked.24230 **Issue 7**

24231 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0063 [320] is applied.

24232 **NAME**

24233 cosh, coshf, coshl ‡hyperbolic cosine functions

24234 **SYNOPSIS**

```
24235 #include <math.h>
24236 double cosh(double x);
24237 float coshf(float x);
24238 long double coshl(long double x);
```

24239 **DESCRIPTION**

24240 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24241 conflict between the requirements described here and the ISO C standard is unintentional. This
 24242 volume of POSIX.1-2017 defers to the ISO C standard.

24243 These functions shall compute the hyperbolic cosine of their argument x .

24244 An application wishing to check for error situations should set *errno* to zero and call
 24245 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 24246 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 24247 zero, an error has occurred.

24248 **RETURN VALUE**

24249 Upon successful completion, these functions shall return the hyperbolic cosine of x .

24250 If the correct value would cause overflow, a range error shall occur and *cosh()*, *coshf()*, and
 24251 *coshl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 24252 respectively.

24253 MX If x is NaN, a NaN shall be returned.

24254 If x is ± 0 , the value 1.0 shall be returned.

24255 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

24256 **ERRORS**

24257 These functions shall fail if:

24258 Range Error The result would cause an overflow.

24259 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 24260 then *errno* shall be set to [ERANGE]. If the integer expression
 24261 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 24262 floating-point exception shall be raised.

24263 **EXAMPLES**

24264 None.

24265 **APPLICATION USAGE**

24266 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 24267 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

24268 **RATIONALE**

24269 None.

24270 **FUTURE DIRECTIONS**

24271 None.

24272 **SEE ALSO**24273 *acosh()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *sinh()*, *tanh()*24274 XBD Section 4.20 (on page 117), **<math.h>**24275 **CHANGE HISTORY**

24276 First released in Issue 1. Derived from Issue 1 of the SVID.

24277 **Issue 5**24278 The DESCRIPTION is updated to indicate how an application should check for an error. This
24279 text was previously published in the APPLICATION USAGE section.24280 **Issue 6**24281 The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.24282 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
24283 revised to align with the ISO/IEC 9899:1999 standard.24284 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
24285 marked.24286 **Issue 7**

24287 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0072 [630] is applied.

24288 **NAME**

24289 cosl †'cosine function

24290 **SYNOPSIS**

24291 #include <math.h>

24292 long double cosl(long double x);

24293 **DESCRIPTION**24294 Refer to *cos()*.

24295 **NAME**

24296 cpow, cpowf, cpowl ‡'complex power functions

24297 **SYNOPSIS**

24298 #include <complex.h>

24299 double complex cpow(double complex x, double complex y);

24300 float complex cpowf(float complex x, float complex y);

24301 long double complex cpowl(long double complex x,

24302 long double complex y);

24303 **DESCRIPTION**24304 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
24305 conflict between the requirements described here and the ISO C standard is unintentional. This
24306 volume of POSIX.1-2017 defers to the ISO C standard.24307 These functions shall compute the complex power function x^y , with a branch cut for the first
24308 parameter along the negative real axis.24309 **RETURN VALUE**

24310 These functions shall return the complex power function value.

24311 **ERRORS**

24312 No errors are defined.

24313 **EXAMPLES**

24314 None.

24315 **APPLICATION USAGE**

24316 None.

24317 **RATIONALE**

24318 None.

24319 **FUTURE DIRECTIONS**

24320 None.

24321 **SEE ALSO**24322 *cabs()*, *csqrt()*

24323 XBD <complex.h>

24324 **CHANGE HISTORY**

24325 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24326 **NAME**

24327 cproj, cprojf, cprojl — complex projection functions

24328 **SYNOPSIS**

```
24329 #include <complex.h>
24330 double complex cproj(double complex z);
24331 float complex cprojf(float complex z);
24332 long double complex cprojl(long double complex z);
```

24333 **DESCRIPTION**

24334 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24335 conflict between the requirements described here and the ISO C standard is unintentional. This
 24336 volume of POSIX.1-2017 defers to the ISO C standard.

24337 These functions shall compute a projection of z onto the Riemann sphere: z projects to z , except
 24338 that all complex infinities (even those with one infinite part and one NaN part) project to
 24339 positive infinity on the real axis. If z has an infinite part, then $cproj(z)$ shall be equivalent to:

```
24340 INFINITY + I * copysign(0.0, cimag(z))
```

24341 **RETURN VALUE**

24342 These functions shall return the value of the projection onto the Riemann sphere.

24343 **ERRORS**

24344 No errors are defined.

24345 **EXAMPLES**

24346 None.

24347 **APPLICATION USAGE**

24348 None.

24349 **RATIONALE**

24350 Two topologies are commonly used in complex mathematics: the complex plane with its
 24351 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
 24352 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
 24353 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
 24354 the complex plane. The $cproj()$ function helps model the Riemann sphere by mapping all
 24355 infinities to one, and should be used just before any operation, especially comparisons, that
 24356 might give spurious results for any of the other infinities. Note that a complex value with one
 24357 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
 24358 infinite, the complex value is infinite independent of the value of the other part. For the same
 24359 reason, $cabs()$ returns an infinity if its argument has an infinite part and a NaN part.

24360 **FUTURE DIRECTIONS**

24361 None.

24362 **SEE ALSO**24363 [carg\(\)](#), [cimag\(\)](#), [conj\(\)](#), [creal\(\)](#)24364 XBD [<complex.h>](#)24365 **CHANGE HISTORY**

24366 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24367 **NAME**

24368 creal, crealf, creall — complex real functions

24369 **SYNOPSIS**

```
24370 #include <complex.h>
24371 double creal(double complex z);
24372 float crealf(float complex z);
24373 long double creall(long double complex z);
```

24374 **DESCRIPTION**

24375 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24376 conflict between the requirements described here and the ISO C standard is unintentional. This
24377 volume of POSIX.1-2017 defers to the ISO C standard.

24378 These functions shall compute the real part of *z*.24379 **RETURN VALUE**

24380 These functions shall return the real part value.

24381 **ERRORS**

24382 No errors are defined.

24383 **EXAMPLES**

24384 None.

24385 **APPLICATION USAGE**24386 For a variable *z* of type **complex**:24387 `z == creal(z) + cimag(z)*I`24388 **RATIONALE**

24389 None.

24390 **FUTURE DIRECTIONS**

24391 None.

24392 **SEE ALSO**24393 *carg()*, *cimag()*, *conj()*, *cproj()*24394 XBD **<complex.h>**24395 **CHANGE HISTORY**

24396 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24397 **NAME**

24398 creat — create a new file or rewrite an existing one

24399 **SYNOPSIS**24400 OH `#include <sys/stat.h>`24401 `#include <fcntl.h>`24402 `int creat(const char *path, mode_t mode);`24403 **DESCRIPTION**24404 The *creat()* function shall behave as if it is implemented as follows:24405 `int creat(const char *path, mode_t mode)`24406 `{`24407 `return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);`24408 `}`24409 **RETURN VALUE**24410 Refer to *open()*.24411 **ERRORS**24412 Refer to *open()*.24413 **EXAMPLES**24414 **Creating a File**24415 The following example creates the file **/tmp/file** with read and write permissions for the file
24416 owner and read permission for group and others. The resulting file descriptor is assigned to the
24417 *fd* variable.24418 `#include <fcntl.h>`24419 `...`24420 `int fd;`24421 `mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;`24422 `char *pathname = "/tmp/file";`24423 `...`24424 `fd = creat(pathname, mode);`24425 `...`24426 **APPLICATION USAGE**

24427 None.

24428 **RATIONALE**24429 The *creat()* function is redundant. Its services are also provided by the *open()* function. It has
24430 been included primarily for historical purposes since many existing applications depend on it. It
24431 is best considered a part of the C binding rather than a function that should be provided in other
24432 languages.24433 **FUTURE DIRECTIONS**

24434 None.

24435 **SEE ALSO**24436 *mknod()*, *open()*24437 XBD *<fcntl.h>*, *<sys/stat.h>*, *<sys/types.h>*

24438 **CHANGE HISTORY**

24439 First released in Issue 1. Derived from Issue 1 of the SVID.

24440 **Issue 6**

24441 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

24442 The following new requirements on POSIX implementations derive from alignment with the
24443 Single UNIX Specification:

24444 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
24445 required for conforming implementations of previous POSIX specifications, it was not
24446 required for UNIX applications.

24447 **Issue 7**

24448 SD5-XSH-ERN-186 is applied.

24449 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0064 [291] is applied.

24450 **NAME**

24451 crypt ‡string encoding function(CRYPT)

24452 **SYNOPSIS**

```
24453 XSI #include <unistd.h>
24454 char *crypt(const char *key, const char *salt);
```

24455 **DESCRIPTION**24456 The *crypt()* function is a string encoding function. The algorithm is implementation-defined.

24457 The *key* argument points to a string to be encoded. The *salt* argument shall be a string of at least
 24458 two bytes in length not including the null character chosen from the set:

```
24459 a b c d e f g h i j k l m n o p q r s t u v w x y z
24460 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
24461 0 1 2 3 4 5 6 7 8 9 . /
```

24462 The first two bytes of this string may be used to perturb the encoding algorithm.

24463 The return value of *crypt()* points to static data that is overwritten by each call.24464 The *crypt()* function need not be thread-safe.24465 **RETURN VALUE**

24466 Upon successful completion, *crypt()* shall return a pointer to the encoded string. The first two
 24467 bytes of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null
 24468 pointer and set *errno* to indicate the error.

24469 **ERRORS**24470 The *crypt()* function shall fail if:

24471 [ENOSYS] The functionality is not supported on this implementation.

24472 **EXAMPLES**24473 **Encoding Passwords**

24474 The following example finds a user database entry matching a particular user name and changes
 24475 the current password to a new password. The *crypt()* function generates an encoded version of
 24476 each password. The first call to *crypt()* produces an encoded version of the old password; that
 24477 encoded password is then compared to the password stored in the user database. The second
 24478 call to *crypt()* encodes the new password before it is stored.

24479 The *putpwent()* function, used in the following example, is not part of POSIX.1-2017.

```
24480 #include <unistd.h>
24481 #include <pwd.h>
24482 #include <string.h>
24483 #include <stdio.h>
24484 ...
24485 int valid_change;
24486 int pfd; /* Integer for file descriptor returned by open(). */
24487 FILE *fpfd; /* File pointer for use in putpwent(). */
24488 struct passwd *p;
24489 char user[100];
24490 char oldpasswd[100];
24491 char newpasswd[100];
24492 char savepasswd[100];
```

```

24493     ...
24494     valid_change = 0;
24495     while ((p = getpwent()) != NULL) {
24496         /* Change entry if found. */
24497         if (strcmp(p->pw_name, user) == 0) {
24498             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
24499                 strcpy(savepasswd, crypt(newpasswd, user));
24500                 p->pw_passwd = savepasswd;
24501                 valid_change = 1;
24502             }
24503             else {
24504                 fprintf(stderr, "Old password is not valid\n");
24505             }
24506         }
24507         /* Put passwd entry into ptmp. */
24508         putpwent(p, fpfd);
24509     }

```

24510 APPLICATION USAGE

24511 The values returned by this function need not be portable among XSI-conformant systems.

24512 Several implementations offer extensions via characters outside of the set specified for the *salt*
 24513 argument for specifying alternative algorithms; while not portable, these extensions may offer
 24514 better security. The use of *crypt()* for anything other than password hashing is not
 24515 recommended.

24516 RATIONALE

24517 None.

24518 FUTURE DIRECTIONS

24519 None.

24520 SEE ALSO

24521 [encrypt\(\)](#), [setkey\(\)](#)

24522 XBD [<unistd.h>](#)

24523 CHANGE HISTORY

24524 First released in Issue 1. Derived from Issue 1 of the SVID.

24525 Issue 5

24526 Normative text previously in the APPLICATION USAGE section is moved to the
 24527 DESCRIPTION.

24528 Issue 7

24529 Austin Group Interpretation 1003.1-2001 #156 is applied.

24530 SD5-XSH-ERN-178 is applied.

24531 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0073 [899] is applied.

24532 **NAME**

24533 csin, csinf, csinl ‡complex sine functions

24534 **SYNOPSIS**

24535 #include <complex.h>

24536 double complex csin(double complex z);

24537 float complex csinf(float complex z);

24538 long double complex csinl(long double complex z);

24539 **DESCRIPTION**

24540 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24541 conflict between the requirements described here and the ISO C standard is unintentional. This
24542 volume of POSIX.1-2017 defers to the ISO C standard.

24543 These functions shall compute the complex sine of z.

24544 **RETURN VALUE**

24545 These functions shall return the complex sine value.

24546 **ERRORS**

24547 No errors are defined.

24548 **EXAMPLES**

24549 None.

24550 **APPLICATION USAGE**

24551 None.

24552 **RATIONALE**

24553 None.

24554 **FUTURE DIRECTIONS**

24555 None.

24556 **SEE ALSO**24557 [casin\(\)](#)24558 XBD [<complex.h>](#)24559 **CHANGE HISTORY**

24560 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24561 **NAME**

24562 csinh, csinhf, csinhl ‡complex hyperbolic sine functions

24563 **SYNOPSIS**

24564 #include <complex.h>

24565 double complex csinh(double complex z);

24566 float complex csinhf(float complex z);

24567 long double complex csinhl(long double complex z);

24568 **DESCRIPTION**

24569 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24570 conflict between the requirements described here and the ISO C standard is unintentional. This
24571 volume of POSIX.1-2017 defers to the ISO C standard.

24572 These functions shall compute the complex hyperbolic sine of z.

24573 **RETURN VALUE**

24574 These functions shall return the complex hyperbolic sine value.

24575 **ERRORS**

24576 No errors are defined.

24577 **EXAMPLES**

24578 None.

24579 **APPLICATION USAGE**

24580 None.

24581 **RATIONALE**

24582 None.

24583 **FUTURE DIRECTIONS**

24584 None.

24585 **SEE ALSO**24586 *casinh()*

24587 XBD <complex.h>

24588 **CHANGE HISTORY**

24589 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24590 **NAME**

24591 csinl ‡complex sine functions

24592 **SYNOPSIS**

24593 #include <complex.h>

24594 long double complex csinl(long double complex z);

24595 **DESCRIPTION**24596 Refer to *csin()*.

24597 **NAME**24598 csqrt, csqrtf, csqrtl *†* complex square root functions24599 **SYNOPSIS**

24600 #include <complex.h>

24601 double complex csqrt(double complex z);

24602 float complex csqrtf(float complex z);

24603 long double complex csqrtl(long double complex z);

24604 **DESCRIPTION**

24605 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24606 conflict between the requirements described here and the ISO C standard is unintentional. This
24607 volume of POSIX.1-2017 defers to the ISO C standard.

24608 These functions shall compute the complex square root of *z*, with a branch cut along the negative
24609 real axis.

24610 **RETURN VALUE**

24611 These functions shall return the complex square root value, in the range of the right half-plane
24612 (including the imaginary axis).

24613 **ERRORS**

24614 No errors are defined.

24615 **EXAMPLES**

24616 None.

24617 **APPLICATION USAGE**

24618 None.

24619 **RATIONALE**

24620 None.

24621 **FUTURE DIRECTIONS**

24622 None.

24623 **SEE ALSO**24624 [cabs\(\)](#), [cpow\(\)](#)24625 XBD [<complex.h>](#)24626 **CHANGE HISTORY**

24627 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24628 **NAME**

24629 ctan, ctanf, ctanl ‡'complex tangent functions

24630 **SYNOPSIS**

24631 #include <complex.h>

24632 double complex ctan(double complex z);

24633 float complex ctanf(float complex z);

24634 long double complex ctanl(long double complex z);

24635 **DESCRIPTION**

24636 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24637 conflict between the requirements described here and the ISO C standard is unintentional. This
24638 volume of POSIX.1-2017 defers to the ISO C standard.

24639 These functions shall compute the complex tangent of *z*.24640 **RETURN VALUE**

24641 These functions shall return the complex tangent value.

24642 **ERRORS**

24643 No errors are defined.

24644 **EXAMPLES**

24645 None.

24646 **APPLICATION USAGE**

24647 None.

24648 **RATIONALE**

24649 None.

24650 **FUTURE DIRECTIONS**

24651 None.

24652 **SEE ALSO**24653 [catan\(\)](#)24654 XBD [<complex.h>](#)24655 **CHANGE HISTORY**

24656 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24657 **NAME**

24658 ctanh, ctanhf, ctanhl ‡complex hyperbolic tangent functions

24659 **SYNOPSIS**

24660 #include <complex.h>

24661 double complex ctanh(double complex z);

24662 float complex ctanhf(float complex z);

24663 long double complex ctanhl(long double complex z);

24664 **DESCRIPTION**24665 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24666 conflict between the requirements described here and the ISO C standard is unintentional. This
24667 volume of POSIX.1-2017 defers to the ISO C standard.24668 These functions shall compute the complex hyperbolic tangent of z .24669 **RETURN VALUE**

24670 These functions shall return the complex hyperbolic tangent value.

24671 **ERRORS**

24672 No errors are defined.

24673 **EXAMPLES**

24674 None.

24675 **APPLICATION USAGE**

24676 None.

24677 **RATIONALE**

24678 None.

24679 **FUTURE DIRECTIONS**

24680 None.

24681 **SEE ALSO**24682 [catanh\(\)](#)24683 XBD [<complex.h>](#)24684 **CHANGE HISTORY**

24685 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

24686 **NAME**

24687 ctanl ‡'complex tangent functions

24688 **SYNOPSIS**

24689 #include <complex.h>

24690 long double complex ctanl(long double complex z);

24691 **DESCRIPTION**24692 Refer to *ctan()*.

24693 **NAME**

24694 ctermid — generate a pathname for the controlling terminal

24695 **SYNOPSIS**

```
24696 CX #include <stdio.h>
24697 char *ctermid(char *s);
```

24698 **DESCRIPTION**

24699 The *ctermid()* function shall generate a string that, when used as a pathname, refers to the
 24700 current controlling terminal for the current process. If *ctermid()* returns a pathname, access to the
 24701 file is not guaranteed.

24702 The *ctermid()* function need not be thread-safe if called with a NULL parameter.

24703 **RETURN VALUE**

24704 If *s* is a null pointer, the string shall be generated in an area that may be static, the address of
 24705 which shall be returned. The application shall not modify the string returned. The returned
 24706 pointer might be invalidated or the string content might be overwritten by a subsequent call to
 24707 *ctermid()*. The returned pointer might also be invalidated if the calling thread is terminated. If *s*
 24708 is not a null pointer, *s* is assumed to point to a character array of at least `L_ctermid` bytes; the
 24709 string is placed in this array and the value of *s* shall be returned. The symbolic constant
 24710 `L_ctermid` is defined in `<stdio.h>`, and shall have a value greater than 0.

24711 The *ctermid()* function shall return an empty string if the pathname that would refer to the
 24712 controlling terminal cannot be determined, or if the function is unsuccessful.

24713 **ERRORS**

24714 No errors are defined.

24715 **EXAMPLES**24716 **Determining the Controlling Terminal for the Current Process**

24717 The following example returns a pointer to a string that identifies the controlling terminal for the
 24718 current process. The pathname for the terminal is stored in the array pointed to by the *ptr*
 24719 argument, which has a size of `L_ctermid` bytes, as indicated by the *term* argument.

```
24720 #include <stdio.h>
24721 ...
24722 char term[L_ctermid];
24723 char *ptr;
24724 ptr = ctermid(term);
```

24725 **APPLICATION USAGE**

24726 The difference between *ctermid()* and *ttyname()* is that *ttyname()* must be handed a file
 24727 descriptor and return a path of the terminal associated with that file descriptor, while *ctermid()*
 24728 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a
 24729 pathname.

24730 **RATIONALE**

24731 `L_ctermid` must be defined appropriately for a given implementation and must be greater than
 24732 zero so that array declarations using it are accepted by the compiler. The value includes the
 24733 terminating null byte.

24734 Conforming applications that use multiple threads cannot call *ctermid()* with NULL as the
 24735 parameter. If *s* is not NULL, the *ctermid()* function generates a string that, when used as a

24736 pathname, refers to the current controlling terminal for the current process. If *s* is NULL, the
24737 return value of *ctermid()* is undefined.

24738 There is no additional burden on the programmer—changing to use a hypothetical thread-safe
24739 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a
24740 buffer. Application code should not assume that the returned string is short, as some
24741 implementations have more than two pathname components before reaching a logical device
24742 name.

24743 **FUTURE DIRECTIONS**

24744 None.

24745 **SEE ALSO**

24746 [ttyname\(\)](#)

24747 XBD [<stdio.h>](#)

24748 **CHANGE HISTORY**

24749 First released in Issue 1. Derived from Issue 1 of the SVID.

24750 **Issue 5**

24751 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

24752 **Issue 6**

24753 The normative text is updated to avoid use of the term “must” for application requirements.

24754 **Issue 7**

24755 Austin Group Interpretation 1003.1-2001 #148 is applied, updating the RATIONALE.

24756 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0065 [75,428] is applied.

24757 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0074 [656] is applied.

24758 **NAME**

24759 ctime, ctime_r †convert a time value to a date and time string

24760 **SYNOPSIS**

```
24761 OB #include <time.h>
24762 char *ctime(const time_t *clock);
24763 OB CX char *ctime_r(const time_t *clock, char *buf);
```

24764 **DESCRIPTION**

24765 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C
 24766 standard. Any conflict between the requirements described here and the ISO C standard is
 24767 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

24768 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds
 24769 since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
24770 asctime(localtime(clock))
```

24771 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 24772 objects: a broken-down time structure and an array of **char**. Execution of any of the functions
 24773 may overwrite the information returned in either of these objects by any of the other functions.

24774 The *ctime()* function need not be thread-safe.

24775 The *ctime_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly
 24776 the same form as *ctime()* and put the string into the array pointed to by *buf* (which shall be at
 24777 least 26 bytes in size) and return *buf*.

24778 Unlike *ctime()*, the *ctime_r()* function is not required to set *tzname*. If *ctime_r()* sets *tzname*, it
 24779 shall also set *daylight* and *timezone*. If *ctime_r()* does not set *tzname*, it shall not set *daylight* and
 24780 shall not set *timezone*.

24781 **RETURN VALUE**

24782 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time
 24783 as an argument.

24784 CX Upon successful completion, *ctime_r()* shall return a pointer to the string pointed to by *buf*.
 24785 When an error is encountered, a null pointer shall be returned.

24786 **ERRORS**

24787 No errors are defined.

24788 **EXAMPLES**

24789 None.

24790 **APPLICATION USAGE**

24791 These functions are included only for compatibility with older implementations. They have
 24792 undefined behavior if the resulting string would be too long, so the use of these functions
 24793 should be discouraged. On implementations that do not detect output string length overflow, it
 24794 is possible to overflow the output buffers in such a way as to cause applications to fail, or
 24795 possible system security violations. Also, these functions do not support localized date and time
 24796 formats. To avoid these problems, applications should use *strftime()* to generate strings from
 24797 broken-down times.

24798 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

24799 The *ctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead of
 24800 possibly using a static data area that may be overwritten by each call.

24801 Attempts to use *ctime()* or *ctime_r()* for times before the Epoch or for times beyond the year 9999
24802 produce undefined results. Refer to *asctime()* (on page 600).

24803 RATIONALE

24804 The standard developers decided to mark the *ctime()* and *ctime_r()* functions obsolescent even
24805 though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C
24806 standard also provides the *strptime()* function which can be used to avoid these problems.

24807 FUTURE DIRECTIONS

24808 These functions may be removed in a future version.

24809 SEE ALSO

24810 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time()*, *utime()*

24811 XBD <[time.h](#)>

24812 CHANGE HISTORY

24813 First released in Issue 1. Derived from Issue 1 of the SVID.

24814 Issue 5

24815 Normative text previously in the APPLICATION USAGE section is moved to the
24816 DESCRIPTION.

24817 The *ctime_r()* function is included for alignment with the POSIX Threads Extension.

24818 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.

24819 Issue 6

24820 Extensions beyond the ISO C standard are marked.

24821 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

24822 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
24823 its avoidance of possibly using a static data area.

24824 Issue 7

24825 Austin Group Interpretation 1003.1-2001 #156 is applied.

24826 SD5-XSH-ERN-25 is applied, updating the APPLICATION USAGE.

24827 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

24828 The *ctime_r()* function is moved from the Thread-Safe Functions option to the Base.

24829 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0066 [321,428] is applied.

24830 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0075 [664] is applied.

24831 **NAME**

24832 daylight ‡daylight savings time flag

24833 **SYNOPSIS**

```
24834 XSI #include <time.h>  
24835 extern int daylight;
```

24836 **DESCRIPTION**

24837 Refer to [tzset\(\)](#).

24838 NAME

24839 dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
24840 dbm_open, dbm_store database functions

24841 SYNOPSIS

```
24842 XSI #include <ndbm.h>
24843 int dbm_clearerr(DBM *db);
24844 void dbm_close(DBM *db);
24845 int dbm_delete(DBM *db, datum key);
24846 int dbm_error(DBM *db);
24847 datum dbm_fetch(DBM *db, datum key);
24848 datum dbm_firstkey(DBM *db);
24849 datum dbm_nextkey(DBM *db);
24850 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
24851 int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

24852 DESCRIPTION

24853 These functions create, access, and modify a database.

24854 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object
24855 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in
24856 the object pointed to by *dptr*.

24857 A database shall be stored in one or two files. When one file is used, the name of the database
24858 file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm_open()*. When
24859 two files are used, the names of the database files shall be formed by appending the suffixes **.dir**
24860 and **.pag** respectively to the *file* argument.

24861 The *dbm_open()* function shall open a database. The *file* argument to the function is the
24862 pathname of the database. The *open_flags* argument has the same meaning as the *flags* argument
24863 of *open()* except that a database opened for write-only access opens the files for read and write
24864 access and the behavior of the **O_APPEND** flag is unspecified. The *file_mode* argument has the
24865 same meaning as the third argument of *open()*.

24866 The *dbm_open()* function need not accept pathnames longer than **{PATH_MAX}-4** bytes
24867 (including the terminating null), or pathnames with a last component longer than
24868 **{NAME_MAX}-4** bytes (excluding the terminating null).

24869 The *dbm_close()* function shall close a database. The application shall ensure that argument *db* is
24870 a pointer to a **dbm** structure that has been returned from a call to *dbm_open()*.

24871 These database functions shall support an internal block size large enough to support
24872 key/content pairs of at least 1 023 bytes.

24873 The *dbm_fetch()* function shall read a record from a database. The argument *db* is a pointer to a
24874 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
24875 **datum** that has been initialized by the application to the value of the key that matches the key of
24876 the record the program is fetching.

24877 The *dbm_store()* function shall write a record to a database. The argument *db* is a pointer to a
24878 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
24879 **datum** that has been initialized by the application to the value of the key that identifies (for
24880 subsequent reading, writing, or deleting) the record the application is writing. The argument
24881 *content* is a **datum** that has been initialized by the application to the value of the record the
24882 program is writing. The argument *store_mode* controls whether *dbm_store()* replaces any pre-
24883 existing record that has the same key that is specified by the *key* argument. The application shall

24884 set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that
 24885 matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be
 24886 replaced with the new record. If the database contains a record that matches the *key* argument
 24887 and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record
 24888 ignored. If the database does not contain a record that matches the *key* argument and *store_mode*
 24889 is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

24890 If the sum of a key/content pair exceeds the internal block size, the result is unspecified.
 24891 Moreover, the application shall ensure that all key/content pairs that hash together fit on a
 24892 single block. The *dbm_store()* function shall return an error in the event that a disk block fills
 24893 with inseparable data.

24894 The *dbm_delete()* function shall delete a record and its key from the database. The argument *db* is
 24895 a pointer to a database structure that has been returned from a call to *dbm_open()*. The argument
 24896 *key* is a **datum** that has been initialized by the application to the value of the key that identifies
 24897 the record the program is deleting.

24898 The *dbm_firstkey()* function shall return the first key in the database. The argument *db* is a
 24899 pointer to a database structure that has been returned from a call to *dbm_open()*.

24900 The *dbm_nextkey()* function shall return the next key in the database. The argument *db* is a
 24901 pointer to a database structure that has been returned from a call to *dbm_open()*. The application
 24902 shall ensure that the *dbm_firstkey()* function is called before calling *dbm_nextkey()*. Subsequent
 24903 calls to *dbm_nextkey()* return the next key until all of the keys in the database have been
 24904 returned.

24905 The *dbm_error()* function shall return the error condition of the database. The argument *db* is a
 24906 pointer to a database structure that has been returned from a call to *dbm_open()*.

24907 The *dbm_clearerr()* function shall clear the error condition of the database. The argument *db* is a
 24908 pointer to a database structure that has been returned from a call to *dbm_open()*.

24909 The *dptr* pointers returned by these functions may point into static storage that may be changed
 24910 by subsequent calls.

24911 These functions need not be thread-safe.

24912 RETURN VALUE

24913 The *dbm_store()* and *dbm_delete()* functions shall return 0 when they succeed and a negative
 24914 value when they fail.

24915 The *dbm_store()* function shall return 1 if it is called with a *flags* value of DBM_INSERT and the
 24916 function finds an existing record with the same key.

24917 The *dbm_error()* function shall return 0 if the error condition is not set and return a non-zero
 24918 value if the error condition is set.

24919 The return value of *dbm_clearerr()* is unspecified.

24920 The *dbm_firstkey()* and *dbm_nextkey()* functions shall return a key **datum**. When the end of the
 24921 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*
 24922 member of the key shall be a null pointer and the error condition of the database shall be set.

24923 The *dbm_fetch()* function shall return a content **datum**. If no record in the database matches the
 24924 key or if an error condition has been detected in the database, the *dptr* member of the content
 24925 shall be a null pointer.

24926 The *dbm_open()* function shall return a pointer to a database structure. If an error is detected
 24927 during the operation, *dbm_open()* shall return a (DBM *)0.

24928 **ERRORS**

24929 No errors are defined.

24930 **EXAMPLES**

24931 None.

24932 **APPLICATION USAGE**

24933 The following code can be used to traverse the database:

24934

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

24935 The *dbm_** functions provided in this library should not be confused in any way with those of a
 24936 general-purpose database management system. These functions do not provide for multiple
 24937 search keys per entry, they do not protect against multi-user access (in other words they do not
 24938 lock records or files), and they do not provide the many other useful database functions that are
 24939 found in more robust database management systems. Creating and updating databases by use of
 24940 these functions is relatively slow because of data copies that occur upon hash collisions. These
 24941 functions are useful for applications requiring fast lookup of relatively static information that is
 24942 to be indexed by a single key.

24943 Note that a strictly conforming application is extremely limited by these functions: since there is
 24944 no way to determine that the keys in use do not all hash to the same value (although that would
 24945 be rare), a strictly conforming application cannot be guaranteed that it can store more than one
 24946 block's worth of data in the database. As long as a key collision does not occur, additional data
 24947 may be stored, but because there is no way to determine whether an error is due to a key
 24948 collision or some other error condition (*dbm_error()* being effectively a Boolean), once an error is
 24949 detected, the application is effectively limited to guessing what the error might be if it wishes to
 24950 continue using these functions.

24951 The *dbm_delete()* function need not physically reclaim file space, although it does make it
 24952 available for reuse by the database.

24953 After calling *dbm_store()* or *dbm_delete()* during a pass through the keys by *dbm_firstkey()* and
 24954 *dbm_nextkey()*, the application should reset the database by calling *dbm_firstkey()* before again
 24955 calling *dbm_nextkey()*. The contents of these files are unspecified and may not be portable.

24956 Applications should take care that database pathname arguments specified to *dbm_open()* are
 24957 not prefixes of unrelated files. This might be done, for example, by placing databases in a
 24958 separate directory.

24959 Since some implementations use three characters for a suffix and others use four characters for a
 24960 suffix, applications should ensure that the maximum portable pathname length passed to
 24961 *dbm_open()* is no greater than {PATH_MAX}-4 bytes, with the last component of the pathname
 24962 no greater than {NAME_MAX}-4 bytes.

24963 **RATIONALE**

24964 Previously the standard required the database to be stored in two files, one file being a directory
 24965 containing a bitmap of keys and having **.dir** as its suffix. The second file containing all data and
 24966 having **.pag** as its suffix. This has been changed not to specify the use of the files and to allow
 24967 newer implementations of the Berkeley DB interface using a single file that have evolved while
 24968 remaining compatible with the application programming interface. The standard developers
 24969 considered removing the specific suffixes altogether but decided to retain them so as not to
 24970 pollute the application file name space more than necessary and to allow for portable backups of
 24971 the database.

24972 **FUTURE DIRECTIONS**

24973 None.

24974 **SEE ALSO**24975 *open()*24976 XBD <*ndbm.h*>24977 **CHANGE HISTORY**

24978 First released in Issue 4, Version 2.

24979 **Issue 5**

24980 Moved from X/OPEN UNIX extension to BASE.

24981 Normative text previously in the APPLICATION USAGE section is moved to the
24982 DESCRIPTION.

24983 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

24984 **Issue 6**

24985 The normative text is updated to avoid use of the term ``must'' for application requirements.

24986 **Issue 7**24987 Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits
24988 newer implementations of the Berkeley DB interface.

24989 Austin Group Interpretation 1003.1-2001 #156 is applied.

24990 **NAME**

24991 difftime — compute the difference between two calendar time values

24992 **SYNOPSIS**

24993 #include <time.h>

24994 double difftime(time_t *time1*, time_t *time0*);24995 **DESCRIPTION**

24996 CX The functionality described on this reference page is aligned with the ISO C standard. Any
24997 conflict between the requirements described here and the ISO C standard is unintentional. This
24998 volume of POSIX.1-2017 defers to the ISO C standard.

24999 The *difftime()* function shall compute the difference between two calendar times (as returned by
25000 *time()*): *time1* − *time0*.

25001 **RETURN VALUE**25002 The *difftime()* function shall return the difference expressed in seconds as a type **double**.25003 **ERRORS**

25004 No errors are defined.

25005 **EXAMPLES**

25006 None.

25007 **APPLICATION USAGE**

25008 None.

25009 **RATIONALE**

25010 None.

25011 **FUTURE DIRECTIONS**

25012 None.

25013 **SEE ALSO**25014 [asctime\(\)](#), [clock\(\)](#), [ctime\(\)](#), [gmtime\(\)](#), [localtime\(\)](#), [mktime\(\)](#), [strftime\(\)](#), [strptime\(\)](#), [time\(\)](#), [utime\(\)](#)25015 XBD [<time.h>](#)25016 **CHANGE HISTORY**

25017 First released in Issue 4. Derived from the ISO C standard.

25018 **NAME**25019 `dirfd` — extract the file descriptor used by a DIR stream25020 **SYNOPSIS**

```
25021 #include <dirent.h>
25022 int dirfd(DIR *dirp);
```

25023 **DESCRIPTION**

25024 The `dirfd()` function shall return a file descriptor referring to the same directory as the `dirp`
 25025 argument. This file descriptor shall be closed by a call to `closedir()`. If any attempt is made to
 25026 close the file descriptor, or to modify the state of the associated description, other than by means
 25027 XSI of `closedir()`, `readdir()`, `readdir_r()`, `rewinddir()`, or `seekdir()`, the behavior is undefined.

25028 **RETURN VALUE**

25029 Upon successful completion, the `dirfd()` function shall return an integer which contains a file
 25030 descriptor for the stream pointed to by `dirp`. Otherwise, it shall return `-1` and shall set `errno` to
 25031 indicate the error.

25032 **ERRORS**

25033 The `dirfd()` function may fail if:

25034 [EINVAL] The `dirp` argument does not refer to a valid directory stream.

25035 **EXAMPLES**

25036 None.

25037 **APPLICATION USAGE**

25038 The `dirfd()` function is intended to be a mechanism by which an application may obtain a file
 25039 descriptor to use for the `fchdir()` function.

25040 **RATIONALE**

25041 This interface was introduced because the Base Definitions volume of POSIX.1-2017 does not
 25042 make public the **DIR** data structure. Applications tend to use the `fchdir()` function on the file
 25043 descriptor returned by this interface, and this has proven useful for security reasons; in
 25044 particular, it is a better technique than others where directory names might change.

25045 The description uses the term “a file descriptor” rather than “the file descriptor”. The
 25046 implication intended is that an implementation that does not use an `fd` for `opendir()` could still
 25047 `open()` the directory to implement the `dirfd()` function. Such a descriptor must be closed later
 25048 during a call to `closedir()`.

25049 If it is necessary to allocate an `fd` to be returned by `dirfd()`, it should be done at the time of a call
 25050 to `opendir()`.

25051 **FUTURE DIRECTIONS**

25052 None.

25053 **SEE ALSO**

25054 [closedir\(\)](#), [fchdir\(\)](#), [fdopendir\(\)](#), [fileno\(\)](#), [open\(\)](#), [readdir\(\)](#)

25055 XBD [<dirent.h>](#)

25056 **CHANGE HISTORY**

25057 First released in Issue 7.

25058 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0067 [422] is applied.

25059 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0076 [572] is applied.

25060 **NAME**

25061 dirname — report the parent directory name of a file pathname

25062 **SYNOPSIS**

```
25063 XSI #include <libgen.h>
25064 char *dirname(char *path);
```

25065 **DESCRIPTION**

25066 The *dirname()* function shall take a pointer to a character string that contains a pathname, and
 25067 return a pointer to a string that is a pathname of the parent directory of that file. The *dirname()*
 25068 function shall not perform pathname resolution; the result shall not be affected by whether or
 25069 not *path* exists or by its file type. Trailing '/' characters in the path that are not also leading '/'
 25070 characters shall not be counted as part of the path.

25071 If *path* does not contain a '/', then *dirname()* shall return a pointer to the string ".". If *path* is a
 25072 null pointer or points to an empty string, *dirname()* shall return a pointer to the string ".".

25073 The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to
 25074 static storage that may then be overwritten by a subsequent call to *dirname()*.

25075 The *dirname()* function need not be thread-safe.

25076 **RETURN VALUE**

25077 The *dirname()* function shall return a pointer to a string as described above.

25078 The *dirname()* function may modify the string pointed to by *path*, and may return a pointer to
 25079 internal storage. The returned pointer might be invalidated or the storage might be overwritten
 25080 by a subsequent call to *dirname()*. The returned pointer might also be invalidated if the calling
 25081 thread is terminated.

25082 **ERRORS**

25083 No errors are defined.

25084 **EXAMPLES**

25085 The following code fragment reads a pathname, changes the current working directory to the
 25086 parent directory, and opens the file.

```
25087 char *path = NULL, *pathcopy;
25088 size_t buflen = 0;
25089 ssize_t linelen = 0;
25090 int fd;

25091 linelen = getline(&path, &buflen, stdin);

25092 path[linelen-1] = 0;
25093 pathcopy = strdup(path);
25094 if (chdir(dirname(pathcopy)) < 0) {
25095     ...
25096 }
25097 if ((fd = open(basename(path), O_RDONLY)) >= 0) {
25098     ...
25099     close (fd);
25100 }
25101 ...
25102 free (pathcopy);
25103 free (path);
```

25104 The EXAMPLES section of the *basename()* function (see *basename()*) includes a table showing
25105 examples of the results of processing several sample pathnames by the *basename()* and *dirname()*
25106 functions and by the *basename* and *dirname* utilities.

25107 APPLICATION USAGE

25108 The *dirname()* and *basename()* functions together yield a complete pathname. The expression
25109 *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

25110 Since the meaning of the leading `"/"` is implementation-defined, *dirname("//foo")* may return
25111 either `"/"` or `'/'` (but nothing else).

25112 RATIONALE

25113 None.

25114 FUTURE DIRECTIONS

25115 None.

25116 SEE ALSO

25117 *basename()*

25118 XBD `<libgen.h>`

25119 XCU *basename*, *dirname*

25120 CHANGE HISTORY

25121 First released in Issue 4, Version 2.

25122 Issue 5

25123 Moved from X/OPEN UNIX extension to BASE.

25124 Normative text previously in the APPLICATION USAGE section is moved to the
25125 DESCRIPTION.

25126 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

25127 Issue 7

25128 Austin Group Interpretation 1003.1-2001 #156 is applied.

25129 The EXAMPLES section is revised.

25130 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0068 [75] is applied.

25131 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0077 [830], XSH/TC2-2008/0078 [612],
25132 XSH/TC2-2008/0079 [830], XSH/TC2-2008/0080 [656], and XSH/TC2-2008/0081 [612] are
25133 applied.

25134 **NAME**

25135 div — compute the quotient and remainder of an integer division

25136 **SYNOPSIS**

25137 #include <stdlib.h>

25138 div_t div(int numer, int denom);

25139 **DESCRIPTION**

25140 CX The functionality described on this reference page is aligned with the ISO C standard. Any
25141 conflict between the requirements described here and the ISO C standard is unintentional. This
25142 volume of POSIX.1-2017 defers to the ISO C standard.

25143 The *div()* function shall compute the quotient and remainder of the division of the numerator
25144 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer
25145 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
25146 represented, the behavior is undefined; otherwise, *quot*denom+rem* shall equal *numer*.

25147 **RETURN VALUE**

25148 The *div()* function shall return a structure of type **div_t**, comprising both the quotient and the
25149 remainder. The structure includes the following members, in any order:

```
25150 int quot; /* quotient */  
25151 int rem; /* remainder */
```

25152 **ERRORS**

25153 No errors are defined.

25154 **EXAMPLES**

25155 None.

25156 **APPLICATION USAGE**

25157 None.

25158 **RATIONALE**

25159 None.

25160 **FUTURE DIRECTIONS**

25161 None.

25162 **SEE ALSO**25163 [ldiv\(\)](#)25164 XBD [<stdlib.h>](#)25165 **CHANGE HISTORY**

25166 First released in Issue 4. Derived from the ISO C standard.

25167 **NAME**

25168 dlclose ‡close a symbol table handle

25169 **SYNOPSIS**

25170 #include <dlfcn.h>

25171 int dlclosel(void *handle);

25172 **DESCRIPTION**25173 The *dlclose()* function shall inform the system that the symbol table handle specified by *handle* is
25174 no longer needed by the application.25175 An application writer may use *dlclose()* to make a statement of intent on the part of the process,
25176 but this statement does not create any requirement upon the implementation. When the symbol
25177 table handle is closed, the implementation may unload the executable object files that were
25178 loaded by *dlopen()* when the symbol table handle was opened and those that were loaded by
25179 *dlsym()* when using the symbol table handle identified by *handle*.25180 Once a symbol table handle has been closed, an application should assume that any symbols
25181 (function identifiers and data object identifiers) made visible using *handle*, are no longer
25182 available to the process.25183 Although a *dlclose()* operation is not required to remove any functions or data objects from the
25184 address space, neither is an implementation prohibited from doing so. The only restriction on
25185 such a removal is that no function nor data object shall be removed to which references have
25186 been relocated, until or unless all such references are removed. For instance, an executable object
25187 file that had been loaded with a *dlopen()* operation specifying the `RTLD_GLOBAL` flag might
25188 provide a target for dynamic relocations performed in the processing of other relocatable
25189 objects—in such environments, an application may assume that no relocation, once made, shall
25190 be undone or remade unless the executable object file containing the relocated object has itself
25191 been removed.25192 **RETURN VALUE**25193 If the referenced symbol table handle was successfully closed, *dlclose()* shall return 0. If *handle*
25194 does not refer to an open symbol table handle or if the symbol table handle could not be closed,
25195 *dlclose()* shall return a non-zero value. More detailed diagnostic information shall be available
25196 through *dlerror()*.25197 **ERRORS**

25198 No errors are defined.

25199 **EXAMPLES**25200 The following example illustrates use of *dlopen()* and *dlclose()*:25201 #include <dlfcn.h>
25202 int eret;
25203 void *mylib;
25204 ...
25205 /* Open a dynamic library and then close it ... */
25206 mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
25207 ...
25208 eret = dlclosel(mylib);
25209 ...

25210 **APPLICATION USAGE**

25211 A conforming application should employ a symbol table handle returned from a *dlopen()*
25212 invocation only within a given scope bracketed by a *dlopen()* operation and the corresponding
25213 *dldclose()* operation. Implementations are free to use reference counting or other techniques such
25214 that multiple calls to *dlopen()* referencing the same executable object file may return a pointer to
25215 the same data object as the symbol table handle.

25216 Implementations are also free to re-use a handle. For these reasons, the value of a handle must
25217 be treated as an opaque data type by the application, used only in calls to *dldsym()* and *dldclose()*.

25218 **RATIONALE**

25219 None.

25220 **FUTURE DIRECTIONS**

25221 None.

25222 **SEE ALSO**

25223 *dlderror()*, *dlopen()*, *dldsym()*

25224 XBD <[dlfcn.h](#)>

25225 **CHANGE HISTORY**

25226 First released in Issue 5.

25227 **Issue 6**

25228 The DESCRIPTION is updated to say that the referenced object is closed “if this is the last
25229 reference to it”.

25230 **Issue 7**

25231 The *dlopen()* function is moved from the XSI option to Base.

25232 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0069 [74] is applied.

25233 **NAME**25234 `dlderror` — get diagnostic information25235 **SYNOPSIS**25236 `#include <dldfcn.h>`25237 `char *dlderror(void);`25238 **DESCRIPTION**

25239 The `dlderror()` function shall return a null-terminated character string (with no trailing
 25240 <newline>) that describes the last error that occurred during dynamic linking processing. If no
 25241 dynamic linking errors have occurred since the last invocation of `dlderror()`, `dlderror()` shall return
 25242 NULL. Thus, invoking `dlderror()` a second time, immediately following a prior invocation, shall
 25243 result in NULL being returned.

25244 It is implementation-defined whether or not the `dlderror()` function is thread-safe. A thread-safe
 25245 implementation shall return only errors that occur on the current thread.

25246 **RETURN VALUE**

25247 If successful, `dlderror()` shall return a null-terminated character string; otherwise, NULL shall be
 25248 returned.

25249 The application shall not modify the string returned. The returned pointer might be invalidated
 25250 or the string content might be overwritten by a subsequent call to `dlderror()` in the same thread (if
 25251 `dlderror()` is thread-safe) or in any thread (if `dlderror()` is not thread-safe). The returned pointer
 25252 might also be invalidated if the calling thread is terminated.

25253 **ERRORS**

25254 No errors are defined.

25255 **EXAMPLES**

25256 The following example prints out the last dynamic linking error:

```

25257 ...
25258 #include <dldfcn.h>
25259 char *errstr;
25260 errstr = dlderror();
25261 if (errstr != NULL)
25262     printf ("A dynamic linking error occurred: (%s)\n", errstr);
25263 ...

```

25264 **APPLICATION USAGE**

25265 Depending on the application environment with respect to asynchronous execution events, such
 25266 as signals or other asynchronous computation sharing the address space, conforming
 25267 applications should use a critical section to retrieve the error pointer and buffer.

25268 **RATIONALE**

25269 None.

25270 **FUTURE DIRECTIONS**

25271 None.

25272 **SEE ALSO**25273 [*dldclose\(\)*](#), [*dldopen\(\)*](#), [*dldsym\(\)*](#)25274 XBD [*<dldfcn.h>*](#)

25275 **CHANGE HISTORY**

25276 First released in Issue 5.

25277 **Issue 6**

25278 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

25279 **Issue 7**

25280 Austin Group Interpretation 1003.1-2001 #156 is applied.

25281 The *dlerror()* function is moved from the XSI option to the Base.25282 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0070 [75], XSH/TC1-2008/0071 [97],
25283 and XSH/TC1-2008/0072 [133] are applied.

25284 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0082 [656] is applied.

25285 **NAME**25286 `dlopen` ‡open a symbol table handle25287 **SYNOPSIS**25288 `#include <dlfcn.h>`25289 `void *dlopen(const char *file, int mode);`25290 **DESCRIPTION**25291 The *dlopen()* function shall make the symbols (function identifiers and data object identifiers) in
25292 the executable object file specified by *file* available to the calling program.25293 The class of executable object files eligible for this operation and the manner of their
25294 construction are implementation-defined, though typically such files are shared libraries or
25295 programs.25296 Implementations may permit the construction of embedded dependencies in executable object
25297 files. In such cases, a *dlopen()* operation shall load those dependencies in addition to the
25298 executable object file specified by *file*. Implementations may also impose specific constraints on
25299 the construction of programs that can employ *dlopen()* and its related services.25300 A successful *dlopen()* shall return a symbol table handle which the caller may use on subsequent
25301 calls to *dlsym()* and *dlclose()*.

25302 The value of this symbol table handle should not be interpreted in any way by the caller.

25303 The *file* argument is used to construct a pathname to the executable object file. If *file* contains a
25304 <slash> character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in
25305 an implementation-defined manner to yield a pathname.25306 If *file* is a null pointer, *dlopen()* shall return a global symbol table handle for the currently
25307 running process image. This symbol table handle shall provide access to the symbols from an
25308 ordered set of executable object files consisting of the original program image file, any
25309 executable object files loaded at program start-up as specified by that process file (for example,
25310 shared libraries), and the set of executable object files loaded using *dlopen()* operations with the
25311 RTLD_GLOBAL flag. As the latter set of executable object files can change during execution, the
25312 set of symbols made available by this symbol table handle can also change dynamically.25313 Only a single copy of an executable object file shall be brought into the address space, even if
25314 *dlopen()* is invoked multiple times in reference to the executable object file, and even if different
25315 pathnames are used to reference the executable object file.25316 The *mode* parameter describes how *dlopen()* shall operate upon *file* with respect to the processing
25317 of relocations and the scope of visibility of the symbols provided within *file*. When an
25318 executable object file is brought into the address space of a process, it may contain references to
25319 symbols whose addresses are not known until the executable object file is loaded.25320 These references shall be relocated before the symbols can be accessed. The *mode* parameter
25321 governs when these relocations take place and may have the following values:25322 **RTLD_LAZY** Relocations shall be performed at an implementation-defined time, ranging
25323 from the time of the *dlopen()* call until the first reference to a given symbol
25324 occurs. Specifying **RTLD_LAZY** should improve performance on
25325 implementations supporting dynamic symbol binding since a process might
25326 not reference all of the symbols in an executable object file. And, for systems
25327 supporting dynamic symbol resolution for normal process execution, this
25328 behavior mimics the normal handling of process execution.

25329 RTLD_NOW All necessary relocations shall be performed when the executable object file is
 25330 first loaded. This may waste some processing if relocations are performed for
 25331 symbols that are never referenced. This behavior may be useful for
 25332 applications that need to know that all symbols referenced during execution
 25333 will be available before *dlopen()* returns.

25334 Any executable object file loaded by *dlopen()* that requires relocations against global symbols
 25335 can reference the symbols in the original process image file, any executable object files loaded at
 25336 program start-up, from the initial process image itself, from any other executable object file
 25337 included in the same *dlopen()* invocation, and any executable object files that were loaded in any
 25338 *dlopen()* invocation and which specified the RTLD_GLOBAL flag. To determine the scope of
 25339 visibility for the symbols loaded with a *dlopen()* invocation, the *mode* parameter should be a
 25340 bitwise-inclusive OR with one of the following values:

25341 RTLD_GLOBAL The executable object file's symbols shall be made available for relocation
 25342 processing of any other executable object file. In addition, symbol lookup
 25343 using *dlopen(NULL,mode)* and an associated *dlsym()* allows executable object
 25344 files loaded with this mode to be searched.

25345 RTLD_LOCAL The executable object file's symbols shall not be made available for relocation
 25346 processing of any other executable object file.

25347 If neither RTLD_GLOBAL nor RTLD_LOCAL is specified, the default behavior is unspecified.

25348 If an executable object file is specified in multiple *dlopen()* invocations, *mode* is interpreted at
 25349 each invocation.

25350 If RTLD_NOW has been specified, all relocations shall have been completed rendering further
 25351 RTLD_NOW operations redundant and any further RTLD_LAZY operations irrelevant.

25352 If RTLD_GLOBAL has been specified, the executable object file shall maintain the
 25353 RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL, as
 25354 long as the executable object file remains in the address space (see *dlclose()*).

25355 Symbols introduced into the process image through calls to *dlopen()* may be used in relocation
 25356 activities. Symbols so introduced may duplicate symbols already defined by the program or
 25357 previous *dlopen()* operations. To resolve the ambiguities such a situation might present, the
 25358 resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two
 25359 such resolution orders are defined: load order and dependency order. Load order establishes an
 25360 ordering among symbol definitions, such that the first definition loaded (including definitions
 25361 from the process image file and any dependent executable object files loaded with it) has priority
 25362 over executable object files added later (by *dlopen()*). Load ordering is used in relocation
 25363 processing. Dependency ordering uses a breadth-first order starting with a given executable
 25364 object file, then all of its dependencies, then any dependents of those, iterating until all
 25365 dependencies are satisfied. With the exception of the global symbol table handle obtained via a
 25366 *dlopen()* operation with a null pointer as the *file* argument, dependency ordering is used by the
 25367 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol table
 25368 handle.

25369 When an executable object file is first made accessible via *dlopen()*, it and its dependent
 25370 executable object files are added in dependency order. Once all the executable object files are
 25371 added, relocations are performed using load order. Note that if an executable object file or its
 25372 dependencies had been previously loaded, the load and dependency orders may yield different
 25373 resolutions.

25374 The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum
 25375 those which are exported as identifiers of global scope by the executable object file. Typically,

25376 such identifiers shall be those that were specified in (for example) C source code as having
25377 **extern** linkage. The precise manner in which an implementation constructs the set of exported
25378 symbols for an executable object file is implementation-defined.

25379 RETURN VALUE

25380 Upon successful completion, *dlopen()* shall return a symbol table handle. If *file* cannot be found,
25381 cannot be opened for reading, is not of an appropriate executable object file format for
25382 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its
25383 symbolic references, *dlopen()* shall return a null pointer. More detailed diagnostic information
25384 shall be available through *dlderror()*.

25385 ERRORS

25386 No errors are defined.

25387 EXAMPLES

25388 Refer to *dlsym()*.

25389 APPLICATION USAGE

25390 None.

25391 RATIONALE

25392 None.

25393 FUTURE DIRECTIONS

25394 None.

25395 SEE ALSO

25396 *dlclose()*, *dlderror()*, *dlsym()*

25397 XBD <dlfcn.h>

25398 CHANGE HISTORY

25399 First released in Issue 5.

25400 Issue 6

25401 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/21 is applied, changing the default
25402 behavior in the DESCRIPTION when neither RTLD_GLOBAL nor RTLD_LOCAL are specified
25403 from implementation-defined to unspecified.

25404 Issue 7

25405 The *dlopen()* function is moved from the XSI option to the Base.

25406 The EXAMPLES section is updated to refer to *dlsym()*.

25407 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0073 [74] is applied.

25408 **NAME**25409 `dlsym` — get the address of a symbol from a symbol table handle25410 **SYNOPSIS**25411 `#include <dlfcn.h>`25412 `void *dlsym(void *restrict handle, const char *restrict name);`25413 **DESCRIPTION**

25414 The `dlsym()` function shall obtain the address of a symbol (a function identifier or a data object
 25415 identifier) defined in the symbol table identified by the `handle` argument. The `handle` argument is
 25416 a symbol table handle returned from a call to `dlopen()` (and which has not since been released by
 25417 a call to `dlclose()`), and `name` is the symbol's name as a character string. The return value from
 25418 `dlsym()`, cast to a pointer to the type of the named symbol, can be used to call (in the case of a
 25419 function) or access the contents of (in the case of a data object) the named symbol.

25420 The `dlsym()` function shall search for the named symbol in the symbol table referenced by `handle`.
 25421 If the symbol table was created with lazy loading (see `RTLD_LAZY` in `dlopen()`), load ordering
 25422 shall be used in `dlsym()` operations to relocate executable object files needed to resolve the
 25423 symbol. The symbol resolution algorithm used shall be dependency order as described in
 25424 `dlopen()`.

25425 The `RTLD_DEFAULT` and `RTLD_NEXT` symbolic constants (which may be defined in
 25426 `<dlfcn.h>`) are reserved for future use as special values that applications may be allowed to use
 25427 for `handle`.

25428 **RETURN VALUE**

25429 Upon successful completion, if `name` names a function identifier, `dlsym()` shall return the address
 25430 of the function converted from type pointer to function to type pointer to **void**; otherwise,
 25431 `dlsym()` shall return the address of the data object associated with the data object identifier
 25432 named by `name` converted from a pointer to the type of the data object to a pointer to **void**. If
 25433 `handle` does not refer to a valid symbol table handle or if the symbol named by `name` cannot be
 25434 found in the symbol table associated with `handle`, `dlsym()` shall return a null pointer.

25435 More detailed diagnostic information shall be available through `dlerror()`.

25436 **ERRORS**

25437 No errors are defined.

25438 **EXAMPLES**

25439 The following example shows how `dlopen()` and `dlsym()` can be used to access either a function
 25440 or a data object. For simplicity, error checking has been omitted.

```

25441 void *handle;
25442 int (*fptr)(int), *iptr, result;
25443 /* open the needed symbol table */
25444 handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);
25445 /* find the address of the function my_function */
25446 fptr = (int (*)(int))dlsym(handle, "my_function");
25447 /* find the address of the data object my_object */
25448 iptr = (int *)dlsym(handle, "my_OBJ");
25449 /* invoke my_function, passing the value of my_OBJ as the parameter */
25450 result = (*fptr)(*iptr);

```

25451 **APPLICATION USAGE**

25452 The following special purpose values for *handle* are reserved for future use and have the
25453 indicated meanings:

25454 **RTLD_DEFAULT** The identifier lookup happens in the normal global scope; that is, a search for
25455 an identifier using *handle* would find the same definition as a direct use of this
25456 identifier in the program code.

25457 **RTLD_NEXT** Specifies the next executable object file after this one that defines *name*. This
25458 one refers to the executable object file containing the invocation of *dlsym()*.
25459 The next executable object file is the one found upon the application of a load
25460 order symbol resolution algorithm (see *dlopen()*). The next symbol is either
25461 one of global scope (because it was introduced as part of the original process
25462 image or because it was added with a *dlopen()* operation including the
25463 **RTLD_GLOBAL** flag), or is in an executable object file that was included in the
25464 same *dlopen()* operation that loaded this one.

25465 The **RTLD_NEXT** flag is useful to navigate an intentionally created hierarchy of multiply-
25466 defined symbols created through interposition. For example, if a program wished to create an
25467 implementation of *malloc()* that embedded some statistics gathering about memory allocations,
25468 such an implementation could use the real *malloc()* definition to perform the memory allocation
25469 ~~and~~ itself only embed the necessary logic to implement the statistics gathering function.

25470 Note that conversion from a **void *** pointer to a function pointer as in:

```
25471 fptr = (int (*)(int))dlsym(handle, "my_function");
```

25472 is not defined by the ISO C standard. This standard requires this conversion to work correctly on
25473 conforming implementations.

25474 **RATIONALE**

25475 None.

25476 **FUTURE DIRECTIONS**

25477 None.

25478 **SEE ALSO**

25479 [dlclose\(\)](#), [dlerror\(\)](#), [dlopen\(\)](#)

25480 XBD [<dlfcn.h>](#)

25481 **CHANGE HISTORY**

25482 First released in Issue 5.

25483 **Issue 6**

25484 The **restrict** keyword is added to the *dlsym()* prototype for alignment with the
25485 ISO/IEC 9899:1999 standard.

25486 The **RTLD_DEFAULT** and **RTLD_NEXT** flags are reserved for future use.

25487 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and
25488 adding text to the **RATIONALE** describing issues related to conversion of pointers to functions
25489 and back again.

25490 **Issue 7**

25491 The *dlsym()* function is moved from the XSI option to the Base.

25492 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0074 [74] is applied.

25493 **NAME**

25494 dprintf ‡'print formatted output

25495 **SYNOPSIS**

```
25496 CX #include <stdio.h>  
25497 int dprintf(int filides, const char *restrict format, ...);
```

25498 **DESCRIPTION**

25499 Refer to *fprintf()*.

25500 **NAME**

25501 `drand48`, `erand48`, `jrand48`, `lcong48`, `lrand48`, `mrnd48`, `nrnd48`, `seed48`, `srand48` ‡ generate
 25502 uniformly distributed pseudo-random numbers

25503 **SYNOPSIS**

```
25504 XSI #include <stdlib.h>
25505
25506 double drand48(void);
25507 double erand48(unsigned short xsubi[3]);
25508 long jrand48(unsigned short xsubi[3]);
25509 void lcong48(unsigned short param[7]);
25510 long lrand48(void);
25511 long mrnd48(void);
25512 long nrnd48(unsigned short xsubi[3]);
25513 unsigned short *seed48(unsigned short seed16v[3]);
25514 void srand48(long seedval);
```

25514 **DESCRIPTION**

25515 This family of functions shall generate pseudo-random numbers using a linear congruential
 25516 algorithm and 48-bit integer arithmetic.

25517 The `drand48()` and `erand48()` functions shall return non-negative, double-precision, floating-
 25518 point values, uniformly distributed over the interval [0.0,1.0).

25519 The `lrand48()` and `nrnd48()` functions shall return non-negative, long integers, uniformly
 25520 distributed over the interval $[0, 2^{31})$.

25521 The `mrnd48()` and `jrand48()` functions shall return signed long integers uniformly distributed
 25522 over the interval $[-2^{31}, 2^{31})$.

25523 The `srand48()`, `seed48()`, and `lcong48()` functions are initialization entry points, one of which
 25524 should be invoked before either `drand48()`, `lrand48()`, or `mrnd48()` is called. (Although it is not
 25525 recommended practice, constant default initializer values shall be supplied automatically if
 25526 `drand48()`, `lrand48()`, or `mrnd48()` is called without a prior call to an initialization entry point.)
 25527 The `erand48()`, `nrnd48()`, and `jrand48()` functions do not require an initialization entry point to
 25528 be called first.

25529 All the routines work by generating a sequence of 48-bit integer values, X_i , according to the
 25530 linear congruential formula:

$$25531 X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

25532 The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless `lcong48()` is invoked,
 25533 the multiplier value a and the addend value c are given by:

$$25534 a = 5DEECE66D_{16} = 273673163155_8$$

$$25535 c = B_{16} = 13_8$$

25536 The value returned by any of the `drand48()`, `erand48()`, `jrand48()`, `lrand48()`, `mrnd48()`, or
 25537 `nrnd48()` functions is computed by first generating the next 48-bit X_i in the sequence. Then the
 25538 appropriate number of bits, according to the type of data item to be returned, are copied from
 25539 the high-order (leftmost) bits of X_i and transformed into the returned value.

25540 The `drand48()`, `lrand48()`, and `mrnd48()` functions store the last 48-bit X_i generated in an
 25541 internal buffer; that is why the application shall ensure that these are initialized prior to being
 25542 invoked. The `erand48()`, `nrnd48()`, and `jrand48()` functions require the calling program to

25543 provide storage for the successive X_i values in the array specified as an argument when the
 25544 functions are invoked. That is why these routines do not have to be initialized; the calling
 25545 program merely has to place the desired initial value of X_i into the array and pass it as an
 25546 argument. By using different arguments, *erand48()*, *nrand48()*, and *rand48()* allow separate
 25547 modules of a large program to generate several *independent* streams of pseudo-random numbers;
 25548 that is, the sequence of numbers in each stream shall *not* depend upon how many times the
 25549 routines are called to generate numbers for the other streams.

25550 The initializer function *srand48()* sets the high-order 32 bits of X_i to the low-order 32 bits
 25551 contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

25552 The initializer function *seed48()* sets the value of X_i to the 48-bit value specified in the argument
 25553 array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16
 25554 bits of X_i are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of X_i are set to the
 25555 low-order 16 bits of *seed16v*[2]. In addition, the previous value of X_i is copied into a 48-bit
 25556 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by
 25557 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is
 25558 to be restarted from a given point at some future time — use the pointer to get at and store the
 25559 last X_i value, and then use this value to reinitialize via *seed48()* when the program is restarted.

25560 The initializer function *lcong48()* allows the user to specify the initial X_i , the multiplier value a ,
 25561 and the addend value c . Argument array elements *param*[0-2] specify X_i , *param*[3-5] specify the
 25562 multiplier a , and *param*[6] specifies the 16-bit addend c . After *lcong48()* is called, a subsequent
 25563 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, a and
 25564 c , specified above.

25565 The *drand48()*, *lrand48()*, and *mrand48()* functions need not be thread-safe.

25566 RETURN VALUE

25567 As described in the DESCRIPTION above.

25568 ERRORS

25569 No errors are defined.

25570 EXAMPLES

25571 None.

25572 APPLICATION USAGE

25573 These functions should be avoided whenever non-trivial requirements (including safety) have to
 25574 be fulfilled.

25575 RATIONALE

25576 None.

25577 FUTURE DIRECTIONS

25578 None.

25579 SEE ALSO

25580 [*initstate\(\)*](#), [*rand\(\)*](#)

25581 XBD [`<stdlib.h>`](#)

25582 CHANGE HISTORY

25583 First released in Issue 1. Derived from Issue 1 of the SVID.

25584 **Issue 5**

25585 A note indicating that the *drand48()*, *lrand48()*, and *rand48()* functions need not be reentrant is
25586 added to the DESCRIPTION.

25587 **Issue 6**

25588 The normative text is updated to avoid use of the term “must” for application requirements.

25589 **Issue 7**

25590 Austin Group Interpretation 1003.1-2001 #156 is applied.

25591 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0083 [743] is applied.

25592 **NAME**

25593 dup, dup2 ‡duplicate an open file descriptor

25594 **SYNOPSIS**

```
25595 #include <unistd.h>
25596 int dup(int filde);
25597 int dup2(int filde, int filde2);
```

25598 **DESCRIPTION**

25599 The *dup*() function provides an alternative interface to the service provided by *fcntl*() using the
 25600 F_DUPFD command. The call *dup*(*filde*) shall be equivalent to:

```
25601 fcntl(filde, F_DUPFD, 0);
```

25602 The *dup2*() function shall cause the file descriptor *filde2* to refer to the same open file
 25603 description as the file descriptor *filde* and to share any locks, and shall return *filde2*. If *filde2* is
 25604 already a valid open file descriptor, it shall be closed first, unless *filde* is equal to *filde2* in which
 25605 case *dup2*() shall return *filde2* without closing it. If the close operation fails to close *filde2*,
 25606 *dup2*() shall return -1 without changing the open file description to which *filde2* refers. If *filde*
 25607 is not a valid file descriptor, *dup2*() shall return -1 and shall not close *filde2*. If *filde2* is less than
 25608 0 or greater than or equal to {OPEN_MAX}, *dup2*() shall return -1 with *errno* set to [EBADF].

25609 Upon successful completion, if *filde* is not equal to *filde2*, the FD_CLOEXEC flag associated
 25610 with *filde2* shall be cleared. If *filde* is equal to *filde2*, the FD_CLOEXEC flag associated with
 25611 *filde2* shall not be changed.

25612 TYM If *filde* refers to a typed memory object, the result of the *dup2*() function is unspecified.

25613 **RETURN VALUE**

25614 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;
 25615 otherwise, -1 shall be returned and *errno* set to indicate the error.

25616 **ERRORS**

25617 The *dup*() function shall fail if:

25618 [EBADF] The *filde* argument is not a valid open file descriptor.

25619 [EMFILE] All file descriptors available to the process are currently open.

25620 The *dup2*() function shall fail if:

25621 [EBADF] The *filde* argument is not a valid open file descriptor or the argument *filde2* is
 25622 negative or greater than or equal to {OPEN_MAX}.

25623 [EINTR] The *dup2*() function was interrupted by a signal.

25624 The *dup2*() function may fail if:

25625 [EIO] An I/O error occurred while attempting to close *filde2*.

25626 **EXAMPLES**25627 **Redirecting Standard Output to a File**

25628 The following example closes standard output for the current processes, re-assigns standard
 25629 output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
25630 #include <unistd.h>
25631 ...
25632 int pfd;
25633 ...
```

```

25634     close(1);
25635     dup(pfd);
25636     close(pfd);
25637     ...

```

25638 **Redirecting Error Messages**

25639 The following example redirects messages from *stderr* to *stdout*.

```

25640     #include <unistd.h>
25641     ...
25642     dup2(1, 2);
25643     ...

```

25644 **APPLICATION USAGE**

25645 Implementations may use file descriptors that must be inherited into child processes for the
 25646 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,
 25647 an application that calls *dup2()* with an arbitrary integer for *fildest* risks non-conforming
 25648 behavior, and *dup2()* can only portably be used to overwrite file descriptor values that the
 25649 application has obtained through explicit actions, or for the three file descriptors corresponding
 25650 to the standard file streams. In order to avoid a race condition of leaking an unintended file
 25651 descriptor into a child process, an application should consider opening all file descriptors with
 25652 the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across *exec*.

25653 **RATIONALE**

25654 The *dup()* function is redundant. Its services are also provided by the *fcntl()* function. It has been
 25655 included in this volume of POSIX.1-2017 primarily for historical reasons, since many existing
 25656 applications use it. On the other hand, the *dup2()* function provides unique services, as no other
 25657 interface is able to atomically replace an existing file descriptor.

25658 The *dup2()* function is not marked obsolescent because it presents a type-safe version of
 25659 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

25660 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

25661 In the description of [EBADF], the case of *fildest* being out of range is covered by the given case of
 25662 *fildest* not being valid. The descriptions for *fildest* and *fildest2* are different because the only kind of
 25663 invalidity that is relevant for *fildest2* is whether it is out of range; that is, it does not matter
 25664 whether *fildest2* refers to an open file when the *dup2()* call is made.

25665 **FUTURE DIRECTIONS**

25666 None.

25667 **SEE ALSO**

25668 *close()*, *fcntl()*, *open()*

25669 XBD <[unistd.h](#)>

25670 **CHANGE HISTORY**

25671 First released in Issue 1. Derived from Issue 1 of the SVID.

25672 **Issue 7**

25673 SD5-XSH-ERN-187 is applied.

25674 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0075 [149,428] and
 25675 XSH/TC1-2008/0076 [149] are applied.

25676 **NAME**

25677 duplocale ‡duplicate a locale object

25678 **SYNOPSIS**

```
25679 CX #include <locale.h>
25680 locale_t duplocale(locale_t locobj);
```

25681 **DESCRIPTION**

25682 The *duplocale()* function shall create a duplicate copy of the locale object referenced by the *locobj*
 25683 argument.

25684 If the *locobj* argument is `LC_GLOBAL_LOCALE`, *duplocale()* shall create a new locale object
 25685 containing a copy of the global locale determined by the *setlocale()* function.

25686 The behavior is undefined if the *locobj* argument is not a valid locale object handle.

25687 **RETURN VALUE**

25688 Upon successful completion, the *duplocale()* function shall return a handle for a new locale
 25689 object. Otherwise, *duplocale()* shall return `(locale_t)0` and set *errno* to indicate the error.

25690 **ERRORS**

25691 The *duplocale()* function shall fail if:

25692 [ENOMEM] There is not enough memory available to create the locale object or load the
 25693 locale data.

25694 **EXAMPLES**25695 **Constructing an Altered Version of an Existing Locale Object**

25696 The following example shows a code fragment to create a slightly altered version of an existing
 25697 locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME*
 25698 category data in the locale object with that from the named locale.

```
25699 #include <locale.h>
25700 ...
25701 locale_t
25702 with_changed_lc_time (locale_t obj, const char *name)
25703 {
25704     locale_t retval = duplocale (obj);
25705     if (retval != (locale_t) 0)
25706     {
25707         locale_t changed = newlocale (LC_TIME_MASK, name, retval);
25708         if (changed == (locale_t) 0)
25709             /* An error occurred. Free all allocated resources. */
25710             freelocale (retval);
25711         retval = changed;
25712     }
25713     return retval;
25714 }
```

25715 APPLICATION USAGE

25716 The use of the *duplocale()* function is recommended for situations where a locale object is being
25717 used in multiple places, and it is possible that the lifetime of the locale object might end before
25718 all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is
25719 needed. This can be achieved by a call to *newlocale()* following the *duplocale()* call.

25720 As with the *newlocale()* function, handles for locale objects created by the *duplocale()* function
25721 should be released by a corresponding call to *freelocale()*.

25722 The *duplocale()* function can also be used in conjunction with *uselocale((locale_t)0)*. This returns
25723 the locale in effect for the calling thread, but can have the value LC_GLOBAL_LOCALE. Passing
25724 LC_GLOBAL_LOCALE to functions such as *isalnum_l()* results in undefined behavior, but
25725 applications can convert it into a usable locale object by using *duplocale()*.

25726 RATIONALE

25727 None.

25728 FUTURE DIRECTIONS

25729 None.

25730 SEE ALSO

25731 *freelocale()*, *newlocale()*, *uselocale()*

25732 XBD <locale.h>

25733 CHANGE HISTORY

25734 First released in Issue 7.

25735 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0077 [283,301], XSH/TC1-2008/0078
25736 [283], and XSH/TC1-2008/0079 [301] are applied.

25737 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0084 [753] is applied.

25738 **NAME**25739 encrypt ‡encoding function **CRYPT**25740 **SYNOPSIS**

```
25741 XSI #include <unistd.h>
25742 void encrypt(char block[64], int edflag);
```

25743 **DESCRIPTION**

25744 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.
 25745 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.

25746 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes
 25747 with values of 0 and 1. The array is modified in place to a similar array using the key set by
 25748 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see
 25749 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to
 25750 [ENOSYS].

25751 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing
 25752 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on
 25753 return, an error has occurred.

25754 The *encrypt()* function need not be thread-safe.

25755 **RETURN VALUE**

25756 The *encrypt()* function shall not return a value.

25757 **ERRORS**

25758 The *encrypt()* function shall fail if:

25759 [ENOSYS] The functionality is not supported on this implementation.

25760 **EXAMPLES**

25761 None.

25762 **APPLICATION USAGE**

25763 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm.

25764 In some environments, decoding might not be implemented. This is related to some Government
 25765 restrictions on encryption and decryption routines. Historical practice has been to ship a
 25766 different version of the encryption library without the decryption feature in the routines
 25767 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

25768 **RATIONALE**

25769 None.

25770 **FUTURE DIRECTIONS**

25771 A future version of the standard may mark this interface as obsolete or remove it altogether.

25772 **SEE ALSO**

25773 *crypt()*, *setkey()*

25774 XBD <unistd.h>

25775 **CHANGE HISTORY**

25776 First released in Issue 1. Derived from Issue 1 of the SVID.

- 25777 **Issue 5**
- 25778 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

- 25779 **Issue 6**
- 25780 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

- 25781 **Issue 7**
- 25782 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 25783 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0085 [899] is applied.

25784 **NAME**

25785 endgrent, getgrent, setgrent — group database entry functions

25786 **SYNOPSIS**

```

25787 XSI      #include <grp.h>
25788         void endgrent(void);
25789         struct group *getgrent(void);
25790         void setgrent(void);

```

25791 **DESCRIPTION**

25792 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an
 25793 entry in the group database. If the group database is not already open, *getgrent()* shall open it
 25794 and return a pointer to a **group** structure containing the first entry in the database. Thereafter, it
 25795 shall return a pointer to a **group** structure containing the next **group** structure in the group
 25796 database, so successive calls may be used to search the entire database.

25797 An implementation that provides extended security controls may impose further
 25798 implementation-defined restrictions on accessing the group database. In particular, the system
 25799 may deny the existence of some or all of the group database entries associated with groups other
 25800 than those groups associated with the caller and may omit users other than the caller from the
 25801 list of members of groups in database entries that are returned.

25802 The *setgrent()* function shall rewind the group database so that the next *getgrent()* call returns
 25803 the first entry, allowing repeated searches.

25804 The *endgrent()* function shall close the group database.

25805 The *setgrent()* and *endgrent()* functions shall not change the setting of *errno* if successful.

25806 On error, the *setgrent()* and *endgrent()* functions shall set *errno* to indicate the error.

25807 Since no value is returned by the *setgrent()* and *endgrent()* functions, an application wishing to
 25808 check for error situations should set *errno* to 0, then call the function, then check *errno*.

25809 These functions need not be thread-safe.

25810 **RETURN VALUE**

25811 On successful completion, *getgrent()* shall return a pointer to a **group** structure. On end-of-file,
 25812 *getgrent()* shall return a null pointer and shall not change the setting of *errno*. On error,
 25813 *getgrent()* shall return a null pointer and *errno* shall be set to indicate the error.

25814 The application shall not modify the structure to which the return value points, nor any storage
 25815 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
 25816 structure, might be invalidated or the structure or the storage areas might be overwritten by a
 25817 subsequent call to *getgrgid()*, *getgrnam()*, or *getgrent()*. The returned pointer, and pointers
 25818 within the structure, might also be invalidated if the calling thread is terminated.

25819 **ERRORS**

25820 These functions may fail if:

25821 [EINTR] A signal was caught during the operation.

25822 [EIO] An I/O error has occurred.

25823 In addition, the *getgrent()* and *setgrent()* functions may fail if:

25824 [EMFILE] All file descriptors available to the process are currently open.

- 25825 [ENFILE] The maximum allowable number of files is currently open in the system.
- 25826 **EXAMPLES**
- 25827 None.
- 25828 **APPLICATION USAGE**
- 25829 These functions are provided due to their historical usage. Applications should avoid
25830 dependencies on fields in the group database, whether the database is a single file, or where in
25831 the file system name space the database resides. Applications should use *getgrnam()* and
25832 *getgrgid()* whenever possible because it avoids these dependencies.
- 25833 **RATIONALE**
- 25834 None.
- 25835 **FUTURE DIRECTIONS**
- 25836 None.
- 25837 **SEE ALSO**
- 25838 *endpwent()*, *getgrgid()*, *getgrnam()*, *getlogin()*
- 25839 XBD <grp.h>
- 25840 **CHANGE HISTORY**
- 25841 First released in Issue 4, Version 2.
- 25842 **Issue 5**
- 25843 Moved from X/OPEN UNIX extension to BASE.
- 25844 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
25845 VALUE section.
- 25846 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.
- 25847 **Issue 6**
- 25848 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 25849 **Issue 7**
- 25850 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 25851 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 25852 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0080 [75] is applied.
- 25853 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0086 [493], XSH/TC2-2008/0087 [656],
25854 and XSH/TC2-2008/0088 [493] are applied.

25855 **NAME**

25856 endhostent, gethostent, sethostent ‡network host database functions

25857 **SYNOPSIS**

```
25858 #include <netdb.h>
25859 void endhostent(void);
25860 struct hostent *gethostent(void);
25861 void sethostent(int stayopen);
```

25862 **DESCRIPTION**

25863 These functions shall retrieve information about hosts. This information is considered to be
25864 stored in a database that can be accessed sequentially or randomly. The implementation of this
25865 database is unspecified.

25866 **Note:** In many cases this database is implemented by the Domain Name System, as documented in
25867 RFC 1034, RFC 1035, and RFC 1886.

25868 The *sethostent()* function shall open a connection to the database and set the next entry for
25869 retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection
25870 shall not be closed by a call to *gethostent()*, and the implementation may maintain an open file
25871 descriptor.

25872 The *gethostent()* function shall read the next entry in the database, opening and closing a
25873 connection to the database as necessary.

25874 Entries shall be returned in **hostent** structures.

25875 The *endhostent()* function shall close the connection to the database, releasing any open file
25876 descriptor.

25877 These functions need not be thread-safe.

25878 **RETURN VALUE**

25879 Upon successful completion, the *gethostent()* function shall return a pointer to a **hostent**
25880 structure if the requested entry was found, and a null pointer if the end of the database was
25881 reached or the requested entry was not found.

25882 The application shall not modify the structure to which the return value points, nor any storage
25883 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
25884 structure, might be invalidated or the structure or the storage areas might be overwritten by a
25885 subsequent call to *gethostent()*. The returned pointer, and pointers within the structure, might
25886 also be invalidated if the calling thread is terminated.

25887 **ERRORS**

25888 No errors are defined for *endhostent()*, *gethostent()*, and *sethostent()*.

25889 **EXAMPLES**

25890 None.

25891 **APPLICATION USAGE**

25892 None.

25893 **RATIONALE**

25894 None.

25895 **FUTURE DIRECTIONS**

25896 None.

25897 **SEE ALSO**25898 [endservent\(\)](#)25899 XBD [<netdb.h>](#)25900 **CHANGE HISTORY**

25901 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25902 **Issue 7**

25903 Austin Group Interpretation 1003.1-2001 #156 is applied.

25904 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0081 [75,428] and
25905 XSH/TC1-2008/0082 [75] are applied.

25906 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0089 [656] is applied.

25907 **NAME**

25908 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent ‡network database functions

25909 **SYNOPSIS**

```
25910 #include <netdb.h>
25911 void endnetent(void);
25912 struct netent *getnetbyaddr(uint32_t net, int type);
25913 struct netent *getnetbyname(const char *name);
25914 struct netent *getnetent(void);
25915 void setnetent(int stayopen);
```

25916 **DESCRIPTION**

25917 These functions shall retrieve information about networks. This information is considered to be
 25918 stored in a database that can be accessed sequentially or randomly. The implementation of this
 25919 database is unspecified.

25920 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-
 25921 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either
 25922 directly, or indirectly through one of the other *getnet**(*)* functions), and the implementation may
 25923 maintain an open file descriptor to the database.

25924 The *getnetent()* function shall read the next entry of the database, opening and closing a
 25925 connection to the database as necessary.

25926 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry
 25927 for which the address family specified by *type* matches the *n_addrtype* member and the network
 25928 number *net* matches the *n_net* member, opening and closing a connection to the database as
 25929 necessary. The *net* argument shall be the network number in host byte order.

25930 The *getnetbyname()* function shall search the database from the beginning and find the first entry
 25931 for which the network name specified by *name* matches the *n_name* member, opening and
 25932 closing a connection to the database as necessary.

25933 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a
 25934 **netent** structure, the members of which shall contain the fields of an entry in the network
 25935 database.

25936 The *endnetent()* function shall close the database, releasing any open file descriptor.

25937 These functions need not be thread-safe.

25938 **RETURN VALUE**

25939 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a
 25940 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the
 25941 database was reached or the requested entry was not found. Otherwise, a null pointer shall be
 25942 returned.

25943 The application shall not modify the structure to which the return value points, nor any storage
 25944 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
 25945 structure, might be invalidated or the structure or the storage areas might be overwritten by a
 25946 subsequent call to *getnetbyaddr()*, *getnetbyname()*, or *getnetent()*. The returned pointer, and
 25947 pointers within the structure, might also be invalidated if the calling thread is terminated.

25948 **ERRORS**

25949 No errors are defined.

25950 **EXAMPLES**

25951 None.

25952 **APPLICATION USAGE**

25953 None.

25954 **RATIONALE**

25955 None.

25956 **FUTURE DIRECTIONS**

25957 None.

25958 **SEE ALSO**25959 XBD <[netdb.h](#)>25960 **CHANGE HISTORY**

25961 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

25962 **Issue 7**

25963 Austin Group Interpretation 1003.1-2001 #156 is applied.

25964 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0083 [75] and XSH/TC1-2008/0084 [75] are applied.

25966 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0090 [656] is applied.

25967 **NAME**

25968 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol
25969 database functions

25970 **SYNOPSIS**

```
25971 #include <netdb.h>
25972 void endprotoent(void);
25973 struct protoent *getprotobyname(const char *name);
25974 struct protoent *getprotobynumber(int proto);
25975 struct protoent *getprotoent(void);
25976 void setprotoent(int stayopen);
```

25977 **DESCRIPTION**

25978 These functions shall retrieve information about protocols. This information is considered to be
25979 stored in a database that can be accessed sequentially or randomly. The implementation of this
25980 database is unspecified.

25981 The *setprotoent()* function shall open a connection to the database, and set the next entry to the
25982 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database
25983 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the
25984 other *getproto**(*)* functions), and the implementation may maintain an open file descriptor for
25985 the database.

25986 The *getprotobyname()* function shall search the database from the beginning and find the first
25987 entry for which the protocol name specified by *name* matches the *p_name* member, opening and
25988 closing a connection to the database as necessary.

25989 The *getprotobynumber()* function shall search the database from the beginning and find the first
25990 entry for which the protocol number specified by *proto* matches the *p_proto* member, opening
25991 and closing a connection to the database as necessary.

25992 The *getprotoent()* function shall read the next entry of the database, opening and closing a
25993 connection to the database as necessary.

25994 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer
25995 to a **protoent** structure, the members of which shall contain the fields of an entry in the network
25996 protocol database.

25997 The *endprotoent()* function shall close the connection to the database, releasing any open file
25998 descriptor.

25999 These functions need not be thread-safe.

26000 **RETURN VALUE**

26001 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a
26002 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of
26003 the database was reached or the requested entry was not found. Otherwise, a null pointer is
26004 returned.

26005 The application shall not modify the structure to which the return value points, nor any storage
26006 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
26007 structure, might be invalidated or the structure or the storage areas might be overwritten by a
26008 subsequent call to *getprotobyname()*, *getprotobynumber()*, or *getprotoent()*. The returned pointer,
26009 and pointers within the structure, might also be invalidated if the calling thread is terminated.

26010 ERRORS

26011 No errors are defined.

26012 EXAMPLES

26013 None.

26014 APPLICATION USAGE

26015 None.

26016 RATIONALE

26017 None.

26018 FUTURE DIRECTIONS

26019 None.

26020 SEE ALSO

26021 XBD <[netdb.h](#)>

26022 CHANGE HISTORY

26023 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

26024 Issue 7

26025 Austin Group Interpretation 1003.1-2001 #156 is applied.

26026 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0085 [75] and XSH/TC1-2008/0086 [75] are applied.

26027
26028 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0091 [656] is applied.

26029 **NAME**

26030 endpwent, getpwent, setpwent ‡user database functions

26031 **SYNOPSIS**

```
26032 XSI #include <pwd.h>
26033 void endpwent(void);
26034 struct passwd *getpwent(void);
26035 void setpwent(void);
```

26036 **DESCRIPTION**

26037 These functions shall retrieve information about users.

26038 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of
 26039 an entry in the user database. Each entry in the user database contains a **passwd** structure. If the
 26040 user database is not already open, *getpwent()* shall open it and return a pointer to a **passwd**
 26041 structure containing the first entry in the database. Thereafter, it shall return a pointer to a
 26042 **passwd** structure containing the next entry in the user database. Successive calls can be used to
 26043 search the entire user database.

26044 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

26045 An implementation that provides extended security controls may impose further
 26046 implementation-defined restrictions on accessing the user database. In particular, the system
 26047 may deny the existence of some or all of the user database entries associated with users other
 26048 than the caller.

26049 The *setpwent()* function shall rewind the user database so that the next *getpwent()* call returns
 26050 the first entry, allowing repeated searches.

26051 The *endpwent()* function shall close the user database.26052 The *setpwent()* and *endpwent()* functions shall not change the setting of *errno* if successful.26053 On error, the *setpwent()* and *endpwent()* functions shall set *errno* to indicate the error.

26054 Since no value is returned by the *setpwent()* and *endpwent()* functions, an application wishing to
 26055 check for error situations should set *errno* to 0, then call the function, then check *errno*.

26056 These functions need not be thread-safe.

26057 **RETURN VALUE**

26058 On successful completion, *getpwent()* shall return a pointer to a **passwd** structure. On end-of-
 26059 file, *getpwent()* shall return a null pointer and shall not change the setting of *errno*. On error,
 26060 *getpwent()* shall return a null pointer and *errno* shall be set to indicate the error.

26061 The application shall not modify the structure to which the return value points, nor any storage
 26062 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
 26063 structure, might be invalidated or the structure or the storage areas might be overwritten by a
 26064 subsequent call to *getpwuid()*, *getpwnam()*, or *getpwent()*. The returned pointer, and pointers
 26065 within the structure, might also be invalidated if the calling thread is terminated.

26066 **ERRORS**

26067 These functions may fail if:

26068 [EINTR] A signal was caught during the operation.

26069 [EIO] An I/O error has occurred.

26070 In addition, *getpwent()* and *setpwent()* may fail if:

- 26071 [EMFILE] All file descriptors available to the process are currently open.
- 26072 [ENFILE] The maximum allowable number of files is currently open in the system.

26073 EXAMPLES

26074 Searching the User Database

26075 The following example uses the *getpwent()* function to get successive entries in the user
 26076 database, returning a pointer to a **passwd** structure that contains information about each user.
 26077 The call to *endpwent()* closes the user database and cleans up.

```
26078 #include <pwd.h>
26079 #include <stdio.h>
26080 void printname(uid_t uid)
26081 {
26082     struct passwd *pwd;
26083     setpwent();
26084     while((pwd = getpwent()) != NULL) {
26085         if (pwd->pw_uid == uid) {
26086             printf("name=%s\n", pwd->pw_name);
26087             break;
26088         }
26089     }
26090     endpwent();
26091 }
```

26092 APPLICATION USAGE

26093 These functions are provided due to their historical usage. Applications should avoid
 26094 dependencies on fields in the password database, whether the database is a single file, or where
 26095 in the file system name space the database resides. Applications should use *getpwuid()*
 26096 whenever possible because it avoids these dependencies.

26097 RATIONALE

26098 None.

26099 FUTURE DIRECTIONS

26100 None.

26101 SEE ALSO

26102 [endgrent\(\)](#), [getlogin\(\)](#), [getpwnam\(\)](#), [getpwuid\(\)](#)

26103 XBD [<pwd.h>](#)

26104 CHANGE HISTORY

26105 First released in Issue 4, Version 2.

26106 Issue 5

26107 Moved from X/OPEN UNIX extension to BASE.

26108 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 26109 VALUE section.

26110 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

26111 **Issue 6**

26112 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

26113 **Issue 7**

26114 Austin Group Interpretation 1003.1-2001 #156 is applied.

26115 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

26116 The EXAMPLES section is revised.

26117 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0087 [75] is applied.

26118 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0092 [493], XSH/TC2-2008/0093 [656],
26119 and XSH/TC2-2008/0094 [493] are applied.

26120 **NAME**

26121 endservent, getservbyname, getservbyport, getservent, setservent ‡network services database
 26122 functions

26123 **SYNOPSIS**

```
26124 #include <netdb.h>
26125 void endservent(void);
26126 struct servent *getservbyname(const char *name, const char *proto);
26127 struct servent *getservbyport(int port, const char *proto);
26128 struct servent *getservent(void);
26129 void setservent(int stayopen);
```

26130 **DESCRIPTION**

26131 These functions shall retrieve information about network services. This information is
 26132 considered to be stored in a database that can be accessed sequentially or randomly. The
 26133 implementation of this database is unspecified.

26134 The *setservent()* function shall open a connection to the database, and set the next entry to the
 26135 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
 26136 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv**(
 26137 functions), and the implementation may maintain an open file descriptor for the database.

26138 The *getservent()* function shall read the next entry of the database, opening and closing a
 26139 connection to the database as necessary.

26140 The *getservbyname()* function shall search the database from the beginning and find the first
 26141 entry for which the service name specified by *name* matches the *s_name* member and the protocol
 26142 name specified by *proto* matches the *s_proto* member, opening and closing a connection to the
 26143 database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be
 26144 matched.

26145 The *getservbyport()* function shall search the database from the beginning and find the first entry
 26146 for which the port specified by *port* matches the *s_port* member and the protocol name specified
 26147 by *proto* matches the *s_proto* member, opening and closing a connection to the database as
 26148 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port*
 26149 argument shall be a value obtained by converting a **uint16_t** in network byte order to **int**.

26150 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a
 26151 **servent** structure, the members of which shall contain the fields of an entry in the network
 26152 services database.

26153 The *endservent()* function shall close the database, releasing any open file descriptor.

26154 These functions need not be thread-safe.

26155 **RETURN VALUE**

26156 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to
 26157 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
 26158 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

26159 The application shall not modify the structure to which the return value points, nor any storage
 26160 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
 26161 structure, might be invalidated or the structure or the storage areas might be overwritten by a
 26162 subsequent call to *getservbyname()*, *getservbyport()*, or *getservent()*. The returned pointer, and
 26163 pointers within the structure, might also be invalidated if the calling thread is terminated.

26164 **ERRORS**

26165 No errors are defined.

26166 **EXAMPLES**

26167 None.

26168 **APPLICATION USAGE**26169 The *port* argument of *getserbyport()* need not be compatible with the port values of all address
26170 families.26171 **RATIONALE**

26172 None.

26173 **FUTURE DIRECTIONS**

26174 None.

26175 **SEE ALSO**26176 *endhostent()*, *endprotoent()*, *htonl()*, *inet_addr()*26177 XBD <[netdb.h](#)>26178 **CHANGE HISTORY**

26179 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

26180 **Issue 7**

26181 Austin Group Interpretation 1003.1-2001 #156 is applied.

26182 SD5-XBD-ERN-14 is applied.

26183 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0088 [75] and XSH/TC1-2008/0089
26184 [75] are applied.

26185 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0095 [656] is applied.

26186 **NAME**

26187 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent ‡'user accounting database
 26188 functions

26189 **SYNOPSIS**

```
26190 XSI #include <utmpx.h>
26191 void endutxent(void);
26192 struct utmpx *getutxent(void);
26193 struct utmpx *getutxid(const struct utmpx *id);
26194 struct utmpx *getutxline(const struct utmpx *line);
26195 struct utmpx *pututxline(const struct utmpx *utmpx);
26196 void setutxent(void);
```

26197 **DESCRIPTION**

26198 These functions shall provide access to the user accounting database.

26199 The *getutxent()* function shall read the next entry from the user accounting database. If the
 26200 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

26201 The *getutxid()* function shall search forward from the current point in the database. If the
 26202 *ut_type* value of the **utmpx** structure pointed to by *id* is *BOOT_TIME*, *OLD_TIME*, or
 26203 *NEW_TIME*, then it shall stop when it finds an entry with a matching *ut_type* value. If the
 26204 *ut_type* value is *INIT_PROCESS*, *LOGIN_PROCESS*, *USER_PROCESS*, or *DEAD_PROCESS*,
 26205 then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member
 26206 matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is
 26207 reached without a match, *getutxid()* shall fail.

26208 The *getutxline()* function shall search forward from the current point in the database until it
 26209 finds an entry of the type *LOGIN_PROCESS* or *USER_PROCESS* which also has a *ut_line* value
 26210 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached
 26211 without a match, *getutxline()* shall fail.

26212 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to
 26213 search for multiple occurrences, the application shall zero out the static data after each success,
 26214 or *getutxline()* may return a pointer to the same **utmpx** structure.

26215 There is one exception to the rule about clearing the structure before further reads are done. The
 26216 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user
 26217 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or
 26218 *getutxline()*, if the application has modified this structure and passed the pointer back to
 26219 *pututxline()*.

26220 For all entries that match a request, the *ut_type* member indicates the type of the entry. Other
 26221 members of the entry shall contain meaningful data based on the value of the *ut_type* member as
 26222 follows:

26223	ut_type Member	Other Members with Meaningful Data
26224	EMPTY	No others
26225	BOOT_TIME	<i>ut_tv</i>
26226	OLD_TIME	<i>ut_tv</i>
26227	NEW_TIME	<i>ut_tv</i>
26228	USER_PROCESS	<i>ut_id</i> , <i>ut_user</i> (login name of the user), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
26229	INIT_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>
26230	LOGIN_PROCESS	<i>ut_id</i> , <i>ut_user</i> (implementation-defined name of the login process), <i>ut_line</i> , <i>ut_pid</i> , <i>ut_tv</i>
26231		
26232	DEAD_PROCESS	<i>ut_id</i> , <i>ut_pid</i> , <i>ut_tv</i>

26233 An implementation that provides extended security controls may impose implementation-
 26234 defined restrictions on accessing the user accounting database. In particular, the system may
 26235 deny the existence of some or all of the user accounting database entries associated with users
 26236 other than the caller.

26237 If the process has appropriate privileges, the *pututxline()* function shall write out the structure
 26238 into the user accounting database. It shall search for a record as if by *getutxid()* that satisfies the
 26239 request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be
 26240 made at the end of the user accounting database.

26241 The *endutxent()* function shall close the user accounting database.

26242 The *setutxent()* function shall reset the input to the beginning of the database. This should be
 26243 done before each search for a new entry if it is desired that the entire database be examined.

26244 These functions need not be thread-safe.

26245 RETURN VALUE

26246 Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a
 26247 **utmpx** structure containing a copy of the requested entry in the user accounting database.
 26248 Otherwise, a null pointer shall be returned.

26249 The return value may point to a static area which is overwritten by a subsequent call to
 26250 *getutxid()* or *getutxline()*.

26251 Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a
 26252 copy of the entry added to the user accounting database. Otherwise, a null pointer shall be
 26253 returned.

26254 The *endutxent()* and *setutxent()* functions shall not return a value.

26255 ERRORS

26256 No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()*
 26257 functions.

26258 The *pututxline()* function may fail if:

26259 [EPERM] The process does not have appropriate privileges.

26260 EXAMPLES

26261 None.

26262 APPLICATION USAGE

26263 The sizes of the arrays in the structure can be found using the *sizeof* operator.

26264 RATIONALE

26265 None.

26266 FUTURE DIRECTIONS

26267 None.

26268 SEE ALSO

26269 XBD [<utmpx.h>](#)

26270 CHANGE HISTORY

26271 First released in Issue 4, Version 2.

26272 Issue 5

26273 Moved from X/OPEN UNIX extension to BASE.

26274 Normative text previously in the APPLICATION USAGE section is moved to the
26275 DESCRIPTION.

26276 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

26277 Issue 6

26278 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

26279 Issue 7

26280 Austin Group Interpretation 1003.1-2001 #156 is applied.

26281 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0090 [213,428] and
26282 XSH/TC1-2008/0091 [213] are applied.

26283 **NAME**26284 **environ** — array of character pointers to the environment strings26285 **SYNOPSIS**26286 extern char ****environ**;26287 **DESCRIPTION**26288 Refer to *exec* and XBD [Chapter 8](#) (on page 173).

26289 **NAME**

26290 erand48 ‡generate uniformly distributed pseudo-random numbers

26291 **SYNOPSIS**

26292 XSI #include <stdlib.h>

26293 double erand48(unsigned short xsubi[3]);

26294 **DESCRIPTION**26295 Refer to *drand48()*.

26296 **NAME**

26297 erf, erff, erfl — error functions

26298 **SYNOPSIS**

```
26299 #include <math.h>
26300 double erf(double x);
26301 float erff(float x);
26302 long double erfl(long double x);
```

26303 **DESCRIPTION**

26304 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26305 conflict between the requirements described here and the ISO C standard is unintentional. This
 26306 volume of POSIX.1-2017 defers to the ISO C standard.

26307 These functions shall compute the error function of their argument x , defined as:

$$26308 \quad -\frac{2}{\pi} \int_0^x e^{-t^2} dt$$

26309 An application wishing to check for error situations should set *errno* to zero and call
 26310 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26311 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26312 zero, an error has occurred.

26313 **RETURN VALUE**

26314 Upon successful completion, these functions shall return the value of the error function.

26315 MX If x is NaN, a NaN shall be returned.

26316 If x is ± 0 , ± 0 shall be returned.

26317 If x is $\pm \text{Inf}$, ± 1 shall be returned.

26318 If the correct value would cause underflow, a range error may occur, and *erf()*, *erff()*, and *erfl()*
 26319 shall return an implementation-defined value no greater in magnitude than DBL_MIN,
 26320 FLT_MIN, and LDBL_MIN, respectively.

26321 MXX If the IEC 60559 Floating-Point option is supported, $2 * x / \text{sqrt}(\pi)$ should be returned.

26322 **ERRORS**

26323 These functions may fail if:

26324 MX **Range Error** The result underflows.

26325 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26326 then *errno* shall be set to [ERANGE]. If the integer expression
 26327 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 26328 floating-point exception shall be raised.

26329 **EXAMPLES**26330 **Computing the Probability for a Normal Variate**

26331 This example shows how to use *erf()* to compute the probability that a normal variate assumes a
26332 value in the range $[x_1, x_2]$ with $x_1 \leq x_2$.

26333 This example uses the constant `M_SQRT1_2` which is part of the XSI option.

```
26334 #include <math.h>
26335
26336 double
26337 Phi(const double x1, const double x2)
26338 {
26339     return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
26340 }
```

26340 **APPLICATION USAGE**

26341 Underflow occurs when $|x| < \text{DBL_MIN} * (\text{sqrt}(\pi)/2)$.

26342 On error, the expressions (*math_errhandling* & `MATH_ERRNO`) and (*math_errhandling* &
26343 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

26344 **RATIONALE**

26345 None.

26346 **FUTURE DIRECTIONS**

26347 None.

26348 **SEE ALSO**

26349 *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

26350 XBD [Section 4.20](#) (on page 117), `<math.h>`

26351 **CHANGE HISTORY**

26352 First released in Issue 1. Derived from Issue 1 of the SVID.

26353 **Issue 5**

26354 The DESCRIPTION is updated to indicate how an application should check for an error. This
26355 text was previously published in the APPLICATION USAGE section.

26356 **Issue 6**

26357 The *erf()* function is no longer marked as an extension.

26358 The *erfc()* function is split out onto its own reference page.

26359 The *erff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

26360 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
26361 revised to align with the ISO/IEC 9899:1999 standard.

26362 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
26363 marked.

26364 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/22 is applied, adding the example to the
26365 EXAMPLES section.

26366 **Issue 7**

26367 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0092 [68] is applied.

26368 **NAME**

26369 erfc, erfcf, erfcl — complementary error functions

26370 **SYNOPSIS**

```
26371 #include <math.h>
26372 double erfc(double x);
26373 float erfcf(float x);
26374 long double erfcl(long double x);
```

26375 **DESCRIPTION**

26376 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26377 conflict between the requirements described here and the ISO C standard is unintentional. This
 26378 volume of POSIX.1-2017 defers to the ISO C standard.

26379 These functions shall compute the complementary error function $1.0 - \text{erf}(x)$.

26380 An application wishing to check for error situations should set *errno* to zero and call
 26381 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 26382 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 26383 zero, an error has occurred.

26384 **RETURN VALUE**

26385 Upon successful completion, these functions shall return the value of the complementary error
 26386 function.

26387 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 26388 MXX and *erfc()*, *erfcf()*, and *erfcl()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not
 26389 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 26390 FLT_MIN, and LDBL_MIN, respectively.

26391 MX If *x* is NaN, a NaN shall be returned.

26392 If *x* is ± 0 , +1 shall be returned.

26393 If *x* is $-\text{Inf}$, +2 shall be returned.

26394 If *x* is $+\text{Inf}$, +0 shall be returned.

26395 MXX If the correct value would cause underflow and is representable, a range error may occur and
 26396 the correct value shall be returned.

26397 **ERRORS**

26398 These functions may fail if:

26399 Range Error The result underflows.

26400 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 26401 then *errno* shall be set to [ERANGE]. If the integer expression
 26402 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 26403 floating-point exception shall be raised.

26404 **EXAMPLES**

26405 None.

26406 **APPLICATION USAGE**26407 The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called
26408 for large *x* and the result subtracted from 1.0.26409 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
26410 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.26411 **RATIONALE**

26412 None.

26413 **FUTURE DIRECTIONS**

26414 None.

26415 **SEE ALSO**26416 *erf()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

26417 XBD Section 4.20 (on page 117), <math.h>

26418 **CHANGE HISTORY**

26419 First released in Issue 1. Derived from Issue 1 of the SVID.

26420 **Issue 5**26421 The DESCRIPTION is updated to indicate how an application should check for an error. This
26422 text was previously published in the APPLICATION USAGE section.26423 **Issue 6**26424 The *erfc()* function is no longer marked as an extension.26425 These functions are split out from the *erf()* reference page.26426 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
26427 revised to align with the ISO/IEC 9899:1999 standard.26428 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
26429 marked.26430 **Issue 7**26431 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0093 [68] and XSH/TC1-2008/0094
26432 [68] are applied.

26433 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0096 [630] is applied.

26434 **NAME**26435 `erff, erfl` — error functions26436 **SYNOPSIS**26437 `#include <math.h>`26438 `float erff(float x);`26439 `long double erfl(long double x);`26440 **DESCRIPTION**26441 Refer to *erf()*.

26442 **NAME**

26443 errno — error return value

26444 **SYNOPSIS**

26445 #include <errno.h>

26446 **DESCRIPTION**26447 The lvalue *errno* is used by many functions to return error values.

26448 Many functions provide an error number in *errno*, which has type **int** and is defined in <**errno.h**>. The value of *errno* shall be defined only after a call to a function for which it is explicitly stated to be set and until it is changed by the next function call or if the application assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by a function's return value. Applications shall obtain the definition of *errno* by the inclusion of <**errno.h**>. No function in this volume of POSIX.1-2017 shall set *errno* to 0. The setting of *errno* after a successful call to a function is unspecified unless the description of that function specifies that *errno* shall not be modified.

26456 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

26459 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant pages.

26461 **RETURN VALUE**

26462 None.

26463 **ERRORS**

26464 None.

26465 **EXAMPLES**

26466 None.

26467 **APPLICATION USAGE**

26468 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard in that they required *errno* to be defined as an external variable, whereas the ISO C standard required only that *errno* be defined as a modifiable lvalue with type **int**.

26471 An application that needs to examine the value of *errno* to determine the error should set it to 0 before a function call, then inspect it before a subsequent function call.

26473 **RATIONALE**

26474 None.

26475 **FUTURE DIRECTIONS**

26476 None.

26477 **SEE ALSO**26478 [Section 2.3](#)26479 XBD <**errno.h**>26480 **CHANGE HISTORY**

26481 First released in Issue 1. Derived from Issue 1 of the SVID.

26482 **Issue 5**

26483 The following sentence is deleted from the DESCRIPTION: “The value of *errno* is 0 at program start-up, but is never set to 0 by any XSI function”.

26485 The DESCRIPTION also no longer states that conforming implementations may support the
26486 declaration:

```
26487 extern int errno;
```

26488 **Issue 6**

26489 Obsolescent text regarding defining *errno* as:

```
26490 extern int errno
```

26491 is removed.

26492 Text regarding no function setting *errno* to zero to indicate an error is changed to no function
26493 shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

26494 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/23 is applied, adding text to the
26495 DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified
26496 unless the description of the function requires that it will not be modified.

26497 **NAME**

26498 environ, execl, execlx, execlp, execv, execve, execvp, fexecve ‡execute a file

26499 **SYNOPSIS**

```
26500 #include <unistd.h>
26501 extern char **environ;
26502 int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
26503 int execlx(const char *path, const char *arg0, ... /*,
26504           (char *)0, char *const envp[]*/);
26505 int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
26506 int execv(const char *path, char *const argv[]);
26507 int execve(const char *path, char *const argv[], char *const envp[]);
26508 int execvp(const char *file, char *const argv[]);
26509 int fexecve(int fd, char *const argv[], char *const envp[]);
```

26510 **DESCRIPTION**

26511 The *exec* family of functions shall replace the current process image with a new process image.
 26512 The new image shall be constructed from a regular, executable file called the *new process image*
 26513 *file*. There shall be no return from a successful *exec*, because the calling process image is overlaid
 26514 by the new process image.

26515 The *fexecve()* function shall be equivalent to the *execve()* function except that the file to be
 26516 executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is
 26517 ignored.

26518 When a C-language program is executed as a result of a call to one of the *exec* family of
 26519 functions, it shall be entered as a C-language function call as follows:

```
26520 int main (int argc, char *argv[]);
```

26521 where *argc* is the argument count and *argv* is an array of character pointers to the arguments
 26522 themselves. In addition, the following variable, which must be declared by the user if it is to be
 26523 used directly:

```
26524 extern char **environ;
```

26525 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*
 26526 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*
 26527 array is not counted in *argc*.

26528 Applications can change the entire environment in a single operation by assigning the *environ*
 26529 variable to point to an array of character pointers to the new environment strings. After
 26530 assigning a new value to *environ*, applications should not rely on the new environment strings
 26531 XSI remaining part of the environment, as a call to *getenv()*, *putenv()*, *setenv()*, *unsetenv()*, or any
 26532 function that is dependent on an environment variable may, on noticing that *environ* has
 26533 changed, copy the environment strings to a new array and assign *environ* to point to it.

26534 Any application that directly modifies the pointers to which the *environ* variable points has
 26535 undefined behavior.

26536 Conforming multi-threaded applications shall not use the *environ* variable to access or modify
 26537 any environment variable while any other thread is concurrently modifying any environment
 26538 variable. A call to any function dependent on any environment variable shall be considered a
 26539 use of the *environ* variable to access that environment variable.

26540 The arguments specified by a program with one of the *exec* functions shall be passed on to the
 26541 new process image in the corresponding *main()* arguments.

26542 The argument *path* points to a pathname that identifies the new process image file.

26543 The argument *file* is used to construct a pathname that identifies the new process image file. If
 26544 the *file* argument contains a <slash> character, the *file* argument shall be used as the pathname
 26545 for this file. Otherwise, the path prefix for this file is obtained by a search of the directories
 26546 passed as the environment variable *PATH* (see XBD Chapter 8, on page 173). If this environment
 26547 variable is not present, the results of the search are implementation-defined.

26548 There are two distinct ways in which the contents of the process image file may cause the
 26549 execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the
 26550 ERRORS section). In the cases where the other members of the *exec* family of functions would
 26551 fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command
 26552 interpreter and the environment of the executed command shall be as if the process invoked the
 26553 *sh* utility using *execl()* as follows:

```
26554 execl(<shell path>, arg0, file, arg1, ..., (char *)0);
```

26555 where <*shell path*> is an unspecified pathname for the *sh* utility, *file* is the process image file, and
 26556 for *execvp()*, where *arg0*, *arg1*, and so on correspond to the values passed to *execvp()* in *argv*[0],
 26557 *argv*[1], and so on.

26558 The arguments represented by *arg0*,... are pointers to null-terminated character strings. These
 26559 strings shall constitute the argument list available to the new process image. The list is
 26560 terminated by a null pointer. The argument *arg0* should point to a filename string that is
 26561 associated with the process being started by one of the *exec* functions.

26562 The argument *argv* is an array of character pointers to null-terminated strings. The application
 26563 shall ensure that the last member of this array is a null pointer. These strings shall constitute the
 26564 argument list available to the new process image. The value in *argv*[0] should point to a filename
 26565 string that is associated with the process being started by one of the *exec* functions.

26566 The argument *envp* is an array of character pointers to null-terminated strings. These strings
 26567 shall constitute the environment for the new process image. The *envp* array is terminated by a
 26568 null pointer.

26569 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the
 26570 environment for the new process image shall be taken from the external variable *environ* in the
 26571 calling process.

26572 The number of bytes available for the new process' combined argument and environment lists is
 26573 {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any
 26574 alignment bytes are included in this total.

26575 File descriptors open in the calling process image shall remain open in the new process image,
 26576 except for those whose close-on-exec flag FD_CLOEXEC is set. For those file descriptors that
 26577 remain open, all attributes of the open file description remain unchanged. For any file descriptor
 26578 that is closed for this reason, file locks are removed as a result of the close as described in *close()*.
 26579 Locks that are not removed by closing of file descriptors remain unchanged.

26580 If file descriptor 0, 1, or 2 would otherwise be closed after a successful call to one of the *exec*
 26581 family of functions, implementations may open an unspecified file for the file descriptor in the
 26582 new process image. If a standard utility or a conforming application is executed with file
 26583 descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the
 26584 environment in which the utility or application is executed shall be deemed non-conforming,
 26585 and consequently the utility or application might not behave as described in this standard.

26586 Directory streams open in the calling process image shall be closed in the new process image.

- 26587 The state of the floating-point environment in the initial thread of the new process image shall
26588 be set to the default.
- 26589 The state of conversion descriptors and message catalog descriptors in the new process image is
26590 undefined.
- 26591 For the new process image, the equivalent of:
- 26592 `setlocale(LC_ALL, "C")`
- 26593 shall be executed at start-up.
- 26594 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default
26595 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by
26596 the calling process image shall be set to be ignored by the new process image. Signals set to be
26597 caught by the calling process image shall be set to the default action in the new process image
26598 (see `<signal.h>`).
- 26599 If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether
26600 the SIGCHLD signal is set to be ignored or to the default action in the new process image.
- 26601 XSI After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and
26602 the SA_ONSTACK flag shall be cleared for all signals.
- 26603 After a successful call to any of the *exec* functions, any functions previously registered by the
26604 `atexit()` or `pthread_atfork()` functions are no longer registered.
- 26605 XSI If the ST_NOSUID bit is set for the file system containing the new process image file, then the
26606 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
26607 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
26608 set, the effective user ID of the new process image shall be set to the user ID of the new process
26609 image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the
26610 effective group ID of the new process image shall be set to the group ID of the new process
26611 image file. The real user ID, real group ID, and supplementary group IDs of the new process
26612 image shall remain the same as those of the calling process image. The effective user ID and
26613 effective group ID of the new process image shall be saved (as the saved set-user-ID and the
26614 saved set-group-ID) for use by `setuid()`.
- 26615 XSI Any shared memory segments attached to the calling process image shall not be attached to the
26616 new process image.
- 26617 Any named semaphores open in the calling process shall be closed as if by appropriate calls to
26618 `sem_close()`.
- 26619 TYM Any blocks of typed memory that were mapped in the calling process are unmapped, as if
26620 `munmap()` was implicitly called to unmap them.
- 26621 ML Memory locks established by the calling process via calls to `mlockall()` or `mlock()` shall be
26622 removed. If locked pages in the address space of the calling process are also mapped into the
26623 address spaces of other processes and are locked by those processes, the locks established by the
26624 other processes shall be unaffected by the call by this process to the *exec* function. If the *exec*
26625 function fails, the effect on memory locks is unspecified.
- 26626 Memory mappings created in the process are unmapped before the address space is rebuilt for
26627 the new process image.
- 26628 SS When the calling process image does not use the SCHED_FIFO, SCHED_RR, or
26629 SCHED_SPORADIC scheduling policies, the scheduling policy and parameters of the new
26630 process image and the initial thread in that new process image are implementation-defined.

26631	PS	When the calling process image uses the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the process policy and scheduling parameter settings shall not be changed by a call to an <i>exec</i> function. The initial thread in the new process image shall inherit the process scheduling policy and parameters. It shall have the default system contention scope, but shall inherit its allocation domain from the calling process image.
26632		
26633	TPS	
26634		
26635		
26636		Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.
26637		
26638	MSG	All open message queue descriptors in the calling process shall be closed, as described in <i>mq_close()</i> .
26639		
26640		Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the <i>exec</i> function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the <i>exec</i> function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the <i>exec</i> function be affected by the presence of outstanding asynchronous I/O operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O may be canceled upon <i>exec</i> , is implementation-defined.
26641		
26642		
26643		
26644		
26645		
26646		
26647	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being <i>exec</i> -ed shall not be reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the process executing the <i>exec</i> function itself.
26648		
26649		
26650		
26651	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.
26652		
26653	OB TRC	If the calling process is being traced, the new process image shall continue to be traced into the same trace stream as the original process image, but the new process image shall not inherit the mapping of trace event names to trace event type identifiers that was defined by calls to the <i>posix_trace_eventid_open()</i> or the <i>posix_trace_trid_eventid_open()</i> functions in the calling process image.
26654		
26655		
26656		
26657		
26658		If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described in the <i>posix_trace_shutdown()</i> function.
26659		
26660		The thread ID of the initial thread in the new process image is unspecified.
26661		The size and location of the stack on which the initial thread in the new process image runs is unspecified.
26662		
26663		The initial thread in the new process image shall have its cancellation type set to PTHREAD_CANCEL_DEFERRED and its cancellation state set to PTHREAD_CANCEL_ENABLED.
26664		
26665		
26666		The initial thread in the new process image shall have all thread-specific data values set to NULL and all thread-specific data keys shall be removed by the call to <i>exec</i> without running destructors.
26667		
26668		
26669		The initial thread in the new process image shall be joinable, as if created with the <i>detachstate</i> attribute set to PTHREAD_CREATE_JOINABLE.
26670		
26671		The new process shall inherit at least the following attributes from the calling process image:
26672	XSI	Nice value (see <i>nice()</i>)

26673	XSI	semadj values (see <i>semop()</i>)
26674		Process ID
26675		Parent process ID
26676		Process group ID
26677		Session membership
26678		Real user ID
26679		Real group ID
26680		Supplementary group IDs
26681		Time left until an alarm clock signal (see <i>alarm()</i>)
26682		Current working directory
26683		Root directory
26684		File mode creation mask (see <i>umask()</i>)
26685	XSI	File size limit (see <i>getrlimit()</i> and <i>setrlimit()</i>)
26686		Process signal mask (see <i>pthread_sigmask()</i>)
26687		Pending signal (see <i>sigpending()</i>)
26688		<i>tms_utime</i> , <i>tms_stime</i> , <i>tms_cutime</i> , and <i>tms_cstime</i> (see <i>times()</i>)
26689	XSI	Resource limits
26690		Controlling terminal
26691	XSI	Interval timers
26692		The initial thread of the new process shall inherit at least the following attributes from the
26693		calling thread:
26694		Signal mask (see <i>sigprocmask()</i> and <i>pthread_sigmask()</i>)
26695		Pending signals (see <i>sigpending()</i>)
26696		All other process attributes defined in this volume of POSIX.1-2017 shall be inherited in the new
26697		process image from the old process image. All other thread attributes defined in this volume of
26698		POSIX.1-2017 shall be inherited in the initial thread in the new process image from the calling
26699		thread in the old process image. The inheritance of process or thread attributes not defined by
26700		this volume of POSIX.1-2017 is implementation-defined.
26701		A call to any <i>exec</i> function from a process with more than one thread shall result in all threads
26702		being terminated and the new executable image being loaded and executed. No destructor
26703		functions or cleanup handlers shall be called.
26704		Upon successful completion, the <i>exec</i> functions shall mark for update the last data access
26705		timestamp of the file. If an <i>exec</i> function failed but was able to locate the process image file,
26706		whether the last data access timestamp is marked for update is unspecified. Should the <i>exec</i>
26707		function succeed, the process image file shall be considered to have been opened with <i>open()</i> .
26708		The corresponding <i>close()</i> shall be considered to occur at a time after this open, but before
26709		process termination or successful completion of a subsequent call to one of the <i>exec</i> functions,
26710		<i>posix_spawn()</i> , or <i>posix_spawnp()</i> . The <i>argv[]</i> and <i>envp[]</i> arrays of pointers and the strings to
26711		which those arrays point shall not be modified by a call to one of the <i>exec</i> functions, except as a
26712		consequence of replacing the process image.

26713 XSI The saved resource limits in the new process image are set to be a copy of the process' corresponding hard and soft limits.
26714

26715 RETURN VALUE

26716 If one of the *exec* functions returns to the calling process image, an error has occurred; the return
26717 value shall be -1 , and *errno* shall be set to indicate the error.

26718 ERRORS

26719 The *exec* functions shall fail if:

26720 [E2BIG] The number of bytes used by the new process image's argument list and
26721 environment list is greater than the system-imposed limit of {ARG_MAX}
26722 bytes.

26723 [EACCES] The new process image file is not a regular file and the implementation does
26724 not support execution of files of its type.

26725 [EINVAL] The new process image file has appropriate privileges and has a recognized
26726 executable binary format, but the system does not support execution of a file
26727 with this format.

26728 The *exec* functions, except for *fexecve()*, shall fail if:

26729 [EACCES] Search permission is denied for a directory listed in the new process image
26730 file's path prefix, or the new process image file denies execution permission.

26731 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*
26732 argument.

26733 [ENAMETOOLONG]

26734 The length of a component of a pathname is longer than {NAME_MAX}.

26735 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an
26736 empty string.

26737 [ENOTDIR] A component of the new process image file's path prefix names an existing file
26738 that is neither a directory nor a symbolic link to a directory, or the new process
26739 image file's pathname contains at least one non- \langle slash \rangle character and ends
26740 with one or more trailing \langle slash \rangle characters and the last pathname
26741 component names an existing file that is neither a directory nor a symbolic
26742 link to a directory.

26743 The *exec* functions, except for *execlp()* and *execvp()*, shall fail if:

26744 [ENOEXEC] The new process image file has the appropriate access permission but has an
26745 unrecognized format.

26746 The *fexecve()* function shall fail if:

26747 [EBADF] The *fd* argument is not a valid file descriptor open for executing.

26748 The *exec* functions may fail if:

26749 [ENOMEM] The new process image requires more memory than is allowed by the
26750 hardware or system-imposed memory management constraints.

26751 The *exec* functions, except for *fexecve()*, may fail if:

26752 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
26753 resolution of the *path* or *file* argument.

26754	[ENAMETOOLONG]	
26755		The length of the <i>path</i> argument or the length of the pathname constructed
26756		from the <i>file</i> argument exceeds {PATH_MAX}, or pathname resolution of a
26757		symbolic link produced an intermediate result with a length that exceeds
26758		{PATH_MAX}.
26759	[ETXTBSY]	
26760		The new process image file is a pure procedure (shared text) file that is
		currently open for writing by some process.

26761 EXAMPLES

26762 Using execl()

26763 The following example executes the *ls* command, specifying the pathname of the executable
 26764 (*/bin/ls*) and using arguments supplied directly to the command to produce single-column
 26765 output.

```
26766 #include <unistd.h>
26767 int ret;
26768 ...
26769 ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

26770 Using execl_e()

26771 The following example is similar to [Using execl\(\)](#). In addition, it specifies the environment for
 26772 the new process image using the *env* argument.

```
26773 #include <unistd.h>
26774 int ret;
26775 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
26776 ...
26777 ret = execl_e ("/bin/ls", "ls", "-l", (char *)0, env);
```

26778 Using execl_p()

26779 The following example searches for the location of the *ls* command among the directories
 26780 specified by the *PATH* environment variable.

```
26781 #include <unistd.h>
26782 int ret;
26783 ...
26784 ret = execl_p ("ls", "ls", "-l", (char *)0);
```

26785 Using execv()

26786 The following example passes arguments to the *ls* command in the *cmd* array.

```
26787 #include <unistd.h>
26788 int ret;
26789 char *cmd[] = { "ls", "-l", (char *)0 };
26790 ...
26791 ret = execv ("/bin/ls", cmd);
```

26792 **Using `execve()`**

26793 The following example passes arguments to the `ls` command in the `cmd` array, and specifies the
26794 environment for the new process image using the `env` argument.

```
26795 #include <unistd.h>
26796
26796 int ret;
26797 char *cmd[] = { "ls", "-l", (char *)0 };
26798 char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
26799 ...
26800 ret = execve ("/bin/ls", cmd, env);
```

26801 **Using `execvp()`**

26802 The following example searches for the location of the `ls` command among the directories
26803 specified by the `PATH` environment variable, and passes arguments to the `ls` command in the
26804 `cmd` array.

```
26805 #include <unistd.h>
26806
26806 int ret;
26807 char *cmd[] = { "ls", "-l", (char *)0 };
26808 ...
26809 ret = execvp ("ls", cmd);
```

26810 **APPLICATION USAGE**

26811 As the state of conversion descriptors and message catalog descriptors in the new process image
26812 is undefined, conforming applications should not rely on their use and should close them prior
26813 to calling one of the `exec` functions.

26814 Applications that require other than the default POSIX locale as the global locale in the new
26815 process image should call `setlocale()` with the appropriate parameters.

26816 When assigning a new value to the `environ` variable, applications should ensure that the
26817 environment to which it will point contains at least the following:

- 26818 1. Any implementation-defined variables required by the implementation to provide a
26819 conforming environment. See the `_CS_V7_ENV` entry in `<unistd.h>` and `confstr()` for
26820 details.
- 26821 2. A value for `PATH` which finds conforming versions of all standard utilities before any
26822 other versions.

26823 The same constraint applies to the `envp` array passed to `execle()` or `execve()`, in order to ensure
26824 that the new process image is invoked in a conforming environment.

26825 Applications should not execute programs with file descriptor 0 not open for reading or with file
26826 descriptor 1 or 2 not open for writing, as this might cause the executed program to misbehave.
26827 In order not to pass on these file descriptors to an executed program, applications should not
26828 just close them but should reopen them on, for example, `/dev/null`. Some implementations may
26829 reopen them automatically, but applications should not rely on this being done.

26830 If an application wants to perform a checksum test of the file being executed before executing it,
26831 the file will need to be opened with read permission to perform the checksum test.

26832 Since execute permission is checked by `fexecve()`, the file description `fd` need not have been
26833 opened with the `O_EXEC` flag. However, if the file to be executed denies read and write
26834 permission for the process preparing to do the `exec`, the only way to provide the `fd` to `fexecve()`

26835 will be to use the O_EXEC flag when opening *fd*. In this case, the application will not be able to
 26836 perform a checksum test since it will not be able to read the contents of the file.

26837 Note that when a file descriptor is opened with O_RDONLY, O_RDWR, or O_WRONLY mode,
 26838 the file descriptor can be used to read, read and write, or write the file, respectively, even if the
 26839 mode of the file changes after the file was opened. Using the O_EXEC open mode is different;
 26840 *fexecve()* will ignore the mode that was used when the file descriptor was opened and the *exec*
 26841 will fail if the mode of the file associated with *fd* does not grant execute permission to the calling
 26842 process at the time *fexecve()* is called.

26843 RATIONALE

26844 Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was
 26845 driven by the same requirement in drafts of the ISO C standard. In fact, historical
 26846 implementations have passed a value of zero when no arguments are supplied to the caller of
 26847 the *exec* functions. This requirement was removed from the ISO C standard and subsequently
 26848 removed from this volume of POSIX.1-2017 as well. The wording, in particular the use of the
 26849 word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to
 26850 the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an
 26851 application. In fact, this is good practice, since many existing applications reference *argv[0]*
 26852 without first checking the value of *argc*.

26853 The requirement on a Strictly Conforming POSIX Application also states that the value passed as
 26854 the first argument be a filename string associated with the process being started. Although some
 26855 existing applications pass a pathname rather than a filename string in some circumstances, a
 26856 filename string is more generally useful, since the common usage of *argv[0]* is in printing
 26857 diagnostics. In some cases the filename passed is not the actual filename of the file; for example,
 26858 many implementations of the *login* utility use a convention of prefixing a <hyphen-minus>
 26859 ('-') to the actual filename, which indicates to the command interpreter being invoked that it is
 26860 a “login shell”.

26861 Also, note that the *test* and *[* utilities require specific strings for the *argv[0]* argument to have
 26862 deterministic behavior across all implementations.

26863 Historically, there have been two ways that implementations can *exec* shell scripts.

26864 One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*
 26865 functions return an [ENOEXEC] error for any file not recognizable as executable, including a
 26866 shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file
 26867 to be a shell script and invoke a known command interpreter to interpret such files. This is now
 26868 required by POSIX.1-2017. These implementations of *execvp()* and *execlp()* only give the
 26869 [ENOEXEC] error in the rare case of a problem with the command interpreter’s executable file.
 26870 Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or
 26871 *execvp()*, although implementations can still give it.

26872 Another way that some historical implementations handle shell scripts is by recognizing the first
 26873 two bytes of the file as the character string “#!” and using the remainder of the first line of the
 26874 file as the name of the command interpreter to execute.

26875 One potential source of confusion noted by the standard developers is over how the contents of
 26876 a process image file affect the behavior of the *exec* family of functions. The following is a
 26877 description of the actions taken:

- 26878 1. If the process image file is a valid executable (in a format that is executable and valid and
 26879 having appropriate privileges) for this system, then the system executes the file.

- 26880 2. If the process image file has appropriate privileges and is in a format that is executable
 26881 but not valid for this system (such as a recognized binary for another architecture), then
 26882 this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
- 26883 3. If the process image file has appropriate privileges but is not otherwise recognized:
- 26884 a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter
 26885 assuming that the process image file is a shell script.
- 26886 b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to
 26887 [ENOEXEC].

26888 Applications that do not require to access their arguments may use the form:

26889 `main(void)`

26890 as specified in the ISO C standard. However, the implementation will always provide the two
 26891 arguments *argc* and *argv*, even if they are not used.

26892 Some implementations provide a third argument to *main()* called *envp*. This is defined as a
 26893 pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments,
 26894 so implementations must support applications written this way. Since this volume of
 26895 POSIX.1-2017 defines the global variable *environ*, which is also provided by historical
 26896 implementations and can be used anywhere that *envp* could be used, there is no functional need
 26897 for the *envp* argument. Applications should use the *getenv()* function rather than accessing the
 26898 environment directly via either *envp* or *environ*. Implementations are required to support the
 26899 two-argument calling sequence, but this does not prohibit an implementation from supporting
 26900 *envp* as an optional third argument.

26901 This volume of POSIX.1-2017 specifies that signals set to SIG_IGN remain set to SIG_IGN, and
 26902 that the new process image inherits the signal mask of the thread that called *exec* in the old
 26903 process image. This is consistent with historical implementations, and it permits some useful
 26904 functionality, such as the *nohup* command. However, it should be noted that many existing
 26905 applications wrongly assume that they start with certain signals set to the default action and/or
 26906 unblocked. In particular, applications written with a simpler signal model that does not include
 26907 blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked
 26908 with some signals blocked. Therefore, it is best not to block or ignore signals across *execs* without
 26909 explicit reason to do so, and especially not to block signals across *execs* of arbitrary (not closely
 26910 cooperating) programs.

26911 The *exec* functions always save the value of the effective user ID and effective group ID of the
 26912 process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of
 26913 the process image file is set.

26914 The statement about *argv[]* and *envp[]* being constants is included to make explicit to future
 26915 writers of language bindings that these objects are completely constant. Due to a limitation of
 26916 the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of
 26917 *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the
 26918 natural choice, given that these functions do not modify either the array of pointers or the
 26919 characters to which the function points, but this would disallow existing correct code. Instead,
 26920 only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src*
 26921 derived from the ISO C standard summarizes the compatibility:

	<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
26922	<i>src:</i>				
26923	char *[]	VALID	‡	VALID	‡
26924	const char *[]		‡ALID	V‡	VALID
26925	char * const []		‡	' ALID	#
26926	const char *const[]		‡	'	VALID‡

26928 Since all existing code has a source type matching the first row, the column that gives the most
 26929 valid combinations is the third column. The only other possibility is the fourth column, but
 26930 using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth
 26931 column cannot be used, because the declaration a non-expert would naturally use would be that
 26932 in the second row.

26933 The ISO C standard and this volume of POSIX.1-2017 do not conflict on the use of *environ*, but
 26934 some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in
 26935 the same way as an entry point (for example, *fork()*), it conforms to both standards. A library can
 26936 contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence and no
 26937 problem ensues. The situation is similar for *environ*: the definition in this volume of
 26938 POSIX.1-2017 is to be used if there is no user-provided *environ* to take precedence. At least three
 26939 implementations are known to exist that solve this problem.

26940 [E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to
 26941 the sum of that and the size of the environment list.

26942 [EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the
 26943 new process image file is corrupted. They are non-conforming.

26944 [EINVAL] This error condition was added to POSIX.1-2017 to allow an implementation
 26945 to detect executable files generated for different architectures, and indicate this
 26946 situation to the application. Historical implementations of shells, *execvp()*, and
 26947 *execlp()* that encounter an [ENOEXEC] error will execute a shell on the
 26948 assumption that the file is a shell script. This will not produce the desired
 26949 effect when the file is a valid executable for a different architecture. An
 26950 implementation may now choose to avoid this problem by returning
 26951 [EINVAL] when a valid executable for a different architecture is encountered.
 26952 Some historical implementations return [EINVAL] to indicate that the *path*
 26953 argument contains a character with the high order bit set. The standard
 26954 developers chose to deviate from historical practice for the following reasons:

- 26955 1. The new utilization of [EINVAL] will provide some measure of utility
 26956 to the user community.
- 26957 2. Historical use of [EINVAL] is not acceptable in an internationalized
 26958 operating environment.

26959 [ENAMETOOLONG] Since the file pathname may be constructed by taking elements in the *PATH*
 26960 variable and putting them together with the filename, the
 26961 [ENAMETOOLONG] error condition could also be reached this way.
 26962

26963 [ETXTBSY] System V returns this error when the executable file is currently open for
 26964 writing by some process. This volume of POSIX.1-2017 neither requires nor
 26965 prohibits this behavior.

26966 Other systems (such as System V) may return [EINTR] from *exec*. This is not addressed by this
 26967 volume of POSIX.1-2017, but implementations may have a window between the call to *exec* and

- 26968 the time that a signal could cause one of the *exec* calls to return with [EINTR].
- 26969 An explicit statement regarding the floating-point environment (as defined in the `<fenv.h>`
26970 header) was added to make it clear that the floating-point environment is set to its default when
26971 a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the
26972 default for other process and thread start-up functions is covered by more generic statements in
26973 their descriptions and can be summarized as follows:
- 26974 *posix_spawn()* Set to default.
- 26975 *fork()* Inherit.
- 26976 *pthread_create()* Inherit.
- 26977 The purpose of the *fexecve()* function is to enable executing a file which has been verified to be
26978 the intended file. It is possible to actively check the file by reading from the file descriptor and be
26979 sure that the file is not exchanged for another between the reading and the execution.
26980 Alternatively, a function like *openat()* can be used to open a file which has been found by
26981 reading the content of a directory using *readdir()*.
- 26982 **FUTURE DIRECTIONS**
- 26983 None.
- 26984 **SEE ALSO**
- 26985 *alarm()*, *atexit()*, *chmod()*, *close()*, *confstr()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*,
26986 *getrlimit()*, *mknod()*, *mmap()*, *nice()*, *open()*, *posix_spawn()*, *posix_trace_create()*,
26987 *posix_trace_event()*, *posix_trace_eventid_equal()*, *pthread_atfork()*, *pthread_sigmask()*, *putenv()*,
26988 *readdir()*, *semop()*, *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*, *sigpending()*, *system()*, *times()*,
26989 *ulimit()*, *umask()*
- 26990 XBD Chapter 8 (on page 173), `<unistd.h>`
- 26991 XCU *test*
- 26992 **CHANGE HISTORY**
- 26993 First released in Issue 1. Derived from Issue 1 of the SVID.
- 26994 **Issue 5**
- 26995 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
26996 Threads Extension.
- 26997 Large File Summit extensions are added.
- 26998 **Issue 6**
- 26999 The following new requirements on POSIX implementations derive from alignment with the
27000 Single UNIX Specification:
- 27001 In the DESCRIPTION, behavior is defined for when the process image file is not a valid
27002 executable.
- 27003 In this version, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective
27004 group ID of the new process image shall be saved (as the saved set-user-ID and the saved
27005 set-group-ID) for use by the *setuid()* function.
- 27006 The [ELOOP] mandatory error condition is added.
- 27007 A second [ENAMETOOLONG] is added as an optional error condition.
- 27008 The [ETXTBSY] optional error condition is added.
- 27009 The following changes were made to align with the IEEE P1003.1a draft standard:

- 27010 The [EINVAL] mandatory error condition is added.
- 27011 The [ELOOP] optional error condition is added.
- 27012 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.
- 27013 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics
27014 for typed memory.
- 27015 The normative text is updated to avoid use of the term “must” for application requirements.
- 27016 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.
- 27017 IEEE PASC Interpretation 1003.1 #132 is applied.
- 27018 The DESCRIPTION is updated to make it explicit that the floating-point environment in the new
27019 process image is set to the default.
- 27020 The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents
27021 of a process image file affect the behavior of the *exec* functions.
- 27022 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to
27023 the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change
27024 addresses a security concern, where implementations may want to reopen file descriptors 0, 1,
27025 and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of
27026 functions.
- 27027 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/24 is applied, applying changes to the
27028 DESCRIPTION, addressing which attributes are inherited by threads, and behavioral
27029 requirements for threads attributes.
- 27030 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/25 is applied, updating text in the
27031 RATIONALE from “the process signal mask be unchanged across an *exec*” to “the new process
27032 image inherits the signal mask of the thread that called *exec* in the old process image”.
- 27033 **Issue 7**
- 27034 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the description of `_CS_V7_ENV`
27035 to the APPLICATION USAGE.
- 27036 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 27037 The *fexecve()* function is added from The Open Group Technical Standard, 2006, Extended API
27038 Set Part 2.
- 27039 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads,
27040 and Timers options is moved to the Base.
- 27041 Changes are made related to support for finegrained timestamps.
- 27042 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0095 [386], XSH/TC1-2008/0096 [167],
27043 XSH/TC1-2008/0097 [291], XSH/TC1-2008/0098 [173], XSH/TC1-2008/0099 [296],
27044 XSH/TC1-2008/00100 [324], XSH/TC1-2008/00101 [296], XSH/TC1-2008/00102 [302],
27045 XSH/TC1-2008/00103 [167], XSH/TC1-2008/00104 [173], and XSH/TC1-2008/00105 [291,429]
27046 are applied.
- 27047 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0097 [584], XSH/TC2-2008/0098 [898],
27048 and XSH/TC2-2008/0099 [734] are applied.

27049 **NAME**

27050 exit — terminate a process

27051 **SYNOPSIS**

27052 #include <stdlib.h>

27053 void exit(int status);

27054 **DESCRIPTION**

27055 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27056 conflict between the requirements described here and the ISO C standard is unintentional. This
 27057 volume of POSIX.1-2017 defers to the ISO C standard.

27058 CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
 27059 the least significant 8 bits (that is, *status* & 0377) shall be available from *wait()* and *waitpid()*; the
 27060 full value shall be available from *waitid()* and in the **siginfo_t** passed to a signal handler for
 27061 SIGCHLD.

27062 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their
 27063 registration, except that a function is called after any previously registered functions that had
 27064 already been called at the time it was registered. Each function is called as many times as it was
 27065 registered. If, during the call to any such function, a call to the *longjmp()* function is made that
 27066 would terminate the call to the registered function, the behavior is undefined.

27067 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall
 27068 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more
 27069 than once, the behavior is undefined.

27070 The *exit()* function shall then flush all open streams with unwritten buffered data and close all
 27071 CX open streams. Finally, the process shall be terminated with the same consequences as described
 27072 in [Consequences of Process Termination](#) (on page 553).

27073 **RETURN VALUE**27074 The *exit()* function does not return.27075 **ERRORS**

27076 No errors are defined.

27077 **EXAMPLES**

27078 None.

27079 **APPLICATION USAGE**

27080 None.

27081 **RATIONALE**27082 See *_Exit()*.27083 **FUTURE DIRECTIONS**

27084 None.

27085 **SEE ALSO**27086 [_Exit\(\)](#), [atexit\(\)](#), [exec](#), [longjmp\(\)](#), [tmpfile\(\)](#), [wait\(\)](#), [waitid\(\)](#)27087 XBD [<stdlib.h>](#)27088 **CHANGE HISTORY**

27089 **Issue 7**

27090 Austin Group Interpretation 1003.1-2001 #031 is applied, separating the `_Exit()` and `_exit()`
27091 functions from the `exit()` function.

27092 Austin Group Interpretation 1003.1-2001 #085 is applied.

27093 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0100 [594] is applied.

27094 **NAME**27095 `exp, expf, expl` †exponential function27096 **SYNOPSIS**

```
27097 #include <math.h>
27098 double exp(double x);
27099 float expf(float x);
27100 long double expl(long double x);
```

27101 **DESCRIPTION**

27102 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27103 conflict between the requirements described here and the ISO C standard is unintentional. This
 27104 volume of POSIX.1-2017 defers to the ISO C standard.

27105 These functions shall compute the base-*e* exponential of *x*.

27106 An application wishing to check for error situations should set *errno* to zero and call
 27107 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 27108 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 27109 zero, an error has occurred.

27110 **RETURN VALUE**

27111 Upon successful completion, these functions shall return the exponential value of *x*.

27112 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*
 27113 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

27114 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 27115 MXX and *exp()*, *expf()*, and *expl()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not
 27116 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 27117 FLT_MIN, and LDBL_MIN, respectively.

27118 MX If *x* is NaN, a NaN shall be returned.

27119 If *x* is ±0, 1 shall be returned.

27120 If *x* is -Inf, +0 shall be returned.

27121 If *x* is +Inf, *x* shall be returned.

27122 MXX If the correct value would cause underflow, and is representable, a range error may occur and
 27123 the correct value shall be returned.

27124 **ERRORS**

27125 These functions shall fail if:

27126 Range Error The result overflows.

27127 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27128 then *errno* shall be set to [ERANGE]. If the integer expression
 27129 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 27130 floating-point exception shall be raised.

27131 These functions may fail if:

27132 Range Error The result underflows.

27133 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27134 then *errno* shall be set to [ERANGE]. If the integer expression
 27135 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 27136 floating-point exception shall be raised.

27137 **EXAMPLES**27138 **Computing the Density of the Standard Normal Distribution**

27139 This function shows an implementation for the density of the standard normal distribution
 27140 using `exp()`. This example uses the constant `M_PI` which is part of the XSI option.

```
27141 #include <math.h>
27142 double
27143 normal_density (double x)
27144 {
27145     return exp(-x*x/2) / sqrt (2*M_PI);
27146 }
```

27147 **APPLICATION USAGE**

27148 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
 27149 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

27150 **RATIONALE**

27151 None.

27152 **FUTURE DIRECTIONS**

27153 None.

27154 **SEE ALSO**

27155 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log\(\)](#)

27156 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

27157 **CHANGE HISTORY**

27158 First released in Issue 1. Derived from Issue 1 of the SVID.

27159 **Issue 5**

27160 The DESCRIPTION is updated to indicate how an application should check for an error. This
 27161 text was previously published in the APPLICATION USAGE section.

27162 **Issue 6**

27163 The `expf()` and `expl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

27164 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 27165 revised to align with the ISO/IEC 9899:1999 standard.

27166 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 27167 marked.

27168 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/26 is applied, adding the example to the
 27169 EXAMPLES section.

27170 **Issue 7**

27171 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0106 [68] and XSH/TC1-2008/0107
 27172 [68] are applied.

27173 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0101 [630] is applied.

27174 **NAME**

27175 exp2, exp2f, exp2l ¶exponential base 2 functions

27176 **SYNOPSIS**

```
27177 #include <math.h>
27178 double exp2(double x);
27179 float exp2f(float x);
27180 long double exp2l(long double x);
```

27181 **DESCRIPTION**

27182 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27183 conflict between the requirements described here and the ISO C standard is unintentional. This
 27184 volume of POSIX.1-2017 defers to the ISO C standard.

27185 These functions shall compute the base-2 exponential of x .

27186 An application wishing to check for error situations should set *errno* to zero and call
 27187 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 27188 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 27189 zero, an error has occurred.

27190 **RETURN VALUE**

27191 Upon successful completion, these functions shall return 2^x .

27192 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and
 27193 *exp2l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 27194 respectively.

27195 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 27196 MXX and *exp2()*, *exp2f()*, and *exp2l()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not
 27197 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 27198 FLT_MIN, and LDBL_MIN, respectively.

27199 MX If x is NaN, a NaN shall be returned.

27200 If x is ± 0 , 1 shall be returned.

27201 If x is $-\text{Inf}$, +0 shall be returned.

27202 If x is $+\text{Inf}$, x shall be returned.

27203 MXX If the correct value would cause underflow, and is representable, a range error may occur and
 27204 the correct value shall be returned.

27205 **ERRORS**

27206 These functions shall fail if:

27207 Range Error The result overflows.

27208 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27209 then *errno* shall be set to [ERANGE]. If the integer expression
 27210 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 27211 floating-point exception shall be raised.

27212 These functions may fail if:

27213 Range Error The result underflows.

27214 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27215 then *errno* shall be set to [ERANGE]. If the integer expression
 27216 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

27217 floating-point exception shall be raised.

27218 **EXAMPLES**

27219 None.

27220 **APPLICATION USAGE**

27221 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
27222 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

27223 **RATIONALE**

27224 None.

27225 **FUTURE DIRECTIONS**

27226 None.

27227 **SEE ALSO**

27228 *exp()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*, *log()*

27229 XBD Section 4.20 (on page 117), <math.h>

27230 **CHANGE HISTORY**

27231 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27232 **Issue 7**

27233 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0108 [68] and XSH/TC1-2008/0109
27234 [68] are applied.

27235 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0102 [630] is applied.

27236 **NAME**

27237 expm1, expm1f, expm1l ‡compute exponential functions

27238 **SYNOPSIS**

```
27239 #include <math.h>
27240 double expm1(double x);
27241 float expm1f(float x);
27242 long double expm1l(long double x);
```

27243 **DESCRIPTION**

27244 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 27245 conflict between the requirements described here and the ISO C standard is unintentional. This
 27246 volume of POSIX.1-2017 defers to the ISO C standard.

27247 These functions shall compute $e^x-1.0$.

27248 An application wishing to check for error situations should set *errno* to zero and call
 27249 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 27250 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 27251 zero, an error has occurred.

27252 **RETURN VALUE**27253 Upon successful completion, these functions return $e^x-1.0$.

27254 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and
 27255 *expm1l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 27256 respectively.

27257 **MX** If *x* is NaN, a NaN shall be returned.27258 If *x* is ± 0 , ± 0 shall be returned.27259 If *x* is $-\text{Inf}$, -1 shall be returned.27260 If *x* is $+\text{Inf}$, *x* shall be returned.27261 If *x* is subnormal, a range error may occur27262 **MXX** and *x* should be returned.

27263 **MX** If *x* is not returned, *expm1()*, *expm1f()*, and *expm1l()* shall return an implementation-defined
 27264 value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

27265 **ERRORS**

27266 These functions shall fail if:

27267 **Range Error** The result overflows.

27268 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27269 then *errno* shall be set to [ERANGE]. If the integer expression
 27270 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 27271 floating-point exception shall be raised.

27272 These functions may fail if:

27273 **MX** **Range Error** The value of *x* is subnormal.

27274 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 27275 then *errno* shall be set to [ERANGE]. If the integer expression
 27276 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 27277 floating-point exception shall be raised.

27278 **EXAMPLES**

27279 None.

27280 **APPLICATION USAGE**27281 The value of $\text{expm1}(x)$ may be more accurate than $\text{exp}(x)-1.0$ for small values of x .27282 The $\text{expm1}()$ and $\text{log1p}()$ functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:27283 $\text{expm1}(n * \text{log1p}(x))/x$ 27284 when x is very small (for example, when calculating small daily interest rates). These functions
27285 also simplify writing accurate inverse hyperbolic functions.27286 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
27287 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.27288 **RATIONALE**

27289 None.

27290 **FUTURE DIRECTIONS**

27291 None.

27292 **SEE ALSO**27293 [exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ilogb\(\)](#), [log1p\(\)](#)27294 XBD [Section 4.20](#) (on page 117), [<math.h>](#)27295 **CHANGE HISTORY**

27296 First released in Issue 4, Version 2.

27297 **Issue 5**

27298 Moved from X/OPEN UNIX extension to BASE.

27299 **Issue 6**27300 The $\text{expm1f}()$ and $\text{expm1l}()$ functions are added for alignment with the ISO/IEC 9899:1999
27301 standard.27302 The $\text{expm1}()$ function is no longer marked as an extension.27303 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
27304 revised to align with the ISO/IEC 9899:1999 standard.27305 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
27306 marked.27307 **Issue 7**

27308 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0110 [68] is applied.

27309 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0103 [630] is applied.

27310 **NAME**

27311 fabs, fabsf, fabsl †'absolute value function

27312 **SYNOPSIS**

```
27313       #include <math.h>
27314       double fabs(double x);
27315       float fabsf(float x);
27316       long double fabsl(long double x);
```

27317 **DESCRIPTION**

27318 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27319 conflict between the requirements described here and the ISO C standard is unintentional. This
 27320 volume of POSIX.1-2017 defers to the ISO C standard.

27321 These functions shall compute the absolute value of their argument x , $|x|$.

27322 **RETURN VALUE**

27323 Upon successful completion, these functions shall return the absolute value of x .

27324 MX If x is NaN, a NaN shall be returned.

27325 If x is ± 0 , $+0$ shall be returned.

27326 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

27327 **ERRORS**

27328 No errors are defined.

27329 **EXAMPLES**27330 **Computing the 1-Norm of a Floating-Point Vector**

27331 This example shows the use of *fabs()* to compute the 1-norm of a vector defined as follows:

```
27332       norm1(v) = |v[0]| + |v[1]| + ... + |v[n-1]|
```

27333 where $|x|$ denotes the absolute value of x , n denotes the vector's dimension $v[i]$ denotes the i -th
 27334 component of v ($0 \leq i < n$).

```
27335       #include <math.h>
27336       double
27337       norm1(const double v[], const int n)
27338       {
27339           int        i;
27340           double   n1_v; /* 1-norm of v */
27341           n1_v = 0;
27342           for (i=0; i<n; i++) {
27343               n1_v += fabs (v[i]);
27344           }
27345           return n1_v;
27346       }
```

27347 **APPLICATION USAGE**

27348 None.

27349 **RATIONALE**

27350 None.

27351 **FUTURE DIRECTIONS**

27352 None.

27353 **SEE ALSO**27354 [isnan\(\)](#)27355 XBD [<math.h>](#)27356 **CHANGE HISTORY**

27357 First released in Issue 1. Derived from Issue 1 of the SVID.

27358 **Issue 5**27359 The DESCRIPTION is updated to indicate how an application should check for an error. This
27360 text was previously published in the APPLICATION USAGE section.27361 **Issue 6**27362 The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.27363 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
27364 revised to align with the ISO/IEC 9899:1999 standard.27365 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
27366 marked.27367 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/27 is applied, adding the example to the
27368 EXAMPLES section.

27369 **NAME**27370 `faccessat` — determine accessibility of a file relative to directory file descriptor27371 **SYNOPSIS**27372 `#include <unistd.h>`27373 `int faccessat(int fd, const char *path, int amode, int flag);`27374 **DESCRIPTION**27375 Refer to [*access\(\)*](#).

27376 **NAME**

27377 fattach † attach a STREAMS-based file descriptor to a file in the file system name space
 27378 (**STREAMS**)

27379 **SYNOPSIS**

```
27380 OB XSR #include <stropts.h>
27381       int fattach(int fildes, const char *path);
```

27382 **DESCRIPTION**

27383 The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively
 27384 associating a pathname with *fildes*. The application shall ensure that the *fildes* argument is a
 27385 valid open file descriptor associated with a STREAMS file. The *path* argument points to a
 27386 pathname of an existing file. The application shall have appropriate privileges or be the owner
 27387 of the file named by *path* and have write permission. A successful call to *fattach()* shall cause all
 27388 pathnames that name the file named by *path* to name the STREAMS file associated with *fildes*,
 27389 until the STREAMS file is detached from the file. A STREAMS file can be attached to more than
 27390 one file and can have several pathnames associated with it.

27391 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user
 27392 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1,
 27393 and the size and device identifier are set to those of the STREAMS file associated with *fildes*. If
 27394 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*),
 27395 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildes*
 27396 refers shall be affected.

27397 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to
 27398 refer to the underlying file.

27399 **RETURN VALUE**

27400 Upon successful completion, *fattach()* shall return 0. Otherwise, -1 shall be returned and *errno*
 27401 set to indicate the error.

27402 **ERRORS**

27403 The *fattach()* function shall fail if:

- | | | |
|----------------------------------|----------------|---|
| 27404
27405
27406 | [EACCES] | Search permission is denied for a component of the path prefix, or the process is the owner of <i>path</i> but does not have write permissions on the file named by <i>path</i> . |
| 27407 | [EBADF] | The <i>fildes</i> argument is not a valid open file descriptor. |
| 27408
27409 | [EBUSY] | The file named by <i>path</i> is currently a mount point or has a STREAMS file attached to it. |
| 27410
27411 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument. |
| 27412
27413 | [ENAMETOOLONG] | The length of a component of a pathname is longer than {NAME_MAX}. |
| 27414 | [ENOENT] | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string. |
| 27415
27416
27417
27418 | [ENOTDIR] | A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>path</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters. |

27419 [EPERM] The effective user ID of the process is not the owner of the file named by *path*
 27420 and the process does not have appropriate privileges.

27421 The *fattach()* function may fail if:

27422 [EINVAL] The *fildev* argument does not refer to a STREAMS file.

27423 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 27424 resolution of the *path* argument.

27425 [ENAMETOOLONG]

27426 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 27427 symbolic link produced an intermediate result with a length that exceeds
 27428 {PATH_MAX}.

27429 [EXDEV] A link to a file on another file system was attempted.

27430 EXAMPLES

27431 Attaching a File Descriptor to a File

27432 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this
 27433 STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-**
 27434 **STREAM**, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file
 27435 handle referring to the STREAMS file associated with *fd*.

```
27436 #include <stropts.h>
27437 ...
27438     int fd;
27439     char *pathname = "/tmp/named-STREAM";
27440     int ret;
27441
27442     ret = fattach(fd, pathname);
```

27442 APPLICATION USAGE

27443 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is
 27444 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the
 27445 replaced file need not be a directory and the replacing file is a STREAMS file.

27446 RATIONALE

27447 The file attributes of a file which has been the subject of an *fattach()* call are specifically set
 27448 because of an artifact of the original implementation. The internal mechanism was the same as
 27449 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when
 27450 applied to a regular file are a little surprising, especially as regards the link count which rigidly
 27451 remains one, even if there were several links originally and despite the fact that all original links
 27452 refer to the STREAM as long as the *fattach()* remains in effect.

27453 FUTURE DIRECTIONS

27454 The *fattach()* function may be removed in a future version.

27455 SEE ALSO

27456 *fdetach()*, *isastream()*

27457 XBD **<stropts.h>**

27458 CHANGE HISTORY

27459 First released in Issue 4, Version 2.

27460 **Issue 5**

27461 Moved from X/OPEN UNIX extension to BASE.

27462 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

27463 **Issue 6**

27464 This function is marked as part of the XSI STREAMS Option Group.

27465 The normative text is updated to avoid use of the term “must” for application requirements.

27466 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
27467 [ELOOP] error condition is added.

27468 **Issue 7**

27469 Austin Group Interpretation 1003.1-2001 #143 is applied.

27470 The *fattach()* function is marked obsolescent.

27471 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
27472 pathname exists but is not a directory or a symbolic link to a directory.

27473 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0111 [146,324] and
27474 XSH/TC1-2008/0112 [291] are applied.

27475 **NAME**

27476 fchdir — change working directory

27477 **SYNOPSIS**

27478 #include <unistd.h>

27479 int fchdir(int *fildev*);27480 **DESCRIPTION**27481 The *fchdir()* function shall be equivalent to *chdir()* except that the directory that is to be the new
27482 current working directory is specified by the file descriptor *fildev*.27483 A conforming application can obtain a file descriptor for a file of type directory using *open()*,
27484 provided that the file status flags and access modes do not contain O_WRONLY or O_RDWR.27485 **RETURN VALUE**27486 Upon successful completion, *fchdir()* shall return 0. Otherwise, it shall return -1 and set *errno* to
27487 indicate the error. On failure the current working directory shall remain unchanged.27488 **ERRORS**27489 The *fchdir()* function shall fail if:27490 [EACCES] Search permission is denied for the directory referenced by *fildev*.27491 [EBADF] The *fildev* argument is not an open file descriptor.27492 [ENOTDIR] The open file descriptor *fildev* does not refer to a directory.27493 The *fchdir()* may fail if:27494 [EINTR] A signal was caught during the execution of *fchdir()*.

27495 [EIO] An I/O error occurred while reading from or writing to the file system.

27496 **EXAMPLES**

27497 None.

27498 **APPLICATION USAGE**

27499 None.

27500 **RATIONALE**

27501 None.

27502 **FUTURE DIRECTIONS**

27503 None.

27504 **SEE ALSO**27505 [chdir\(\)](#), [dirfd\(\)](#)27506 XBD [<unistd.h>](#)27507 **CHANGE HISTORY**

27508 First released in Issue 4, Version 2.

27509 **Issue 5**

27510 Moved from X/OPEN UNIX extension to BASE.

27511 **Issue 7**27512 The *fchdir()* function is moved from the XSI option to the Base.

27513 **NAME**27514 `fchmod` ‡'change mode of a file27515 **SYNOPSIS**

```
27516 #include <sys/stat.h>
27517 int fchmod(int fildev, mode_t mode);
```

27518 **DESCRIPTION**

27519 The `fchmod()` function shall be equivalent to `chmod()` except that the file whose permissions are
 27520 changed is specified by the file descriptor *fildev*.

27521 SHM If *fildev* references a shared memory object, the `fchmod()` function need only affect the `S_IRUSR`,
 27522 `S_IWUSR`, `S_IRGRP`, `S_IWGRP`, `S_IROTH`, and `S_IWOTH` file permission bits.

27523 TYM If *fildev* references a typed memory object, the behavior of `fchmod()` is unspecified.

27524 If *fildev* refers to a socket, the behavior of `fchmod()` is unspecified.

27525 OB XSR If *fildev* refers to a STREAM (which is `fattach()`-ed into the file system name space) the call
 27526 returns successfully, doing nothing.

27527 **RETURN VALUE**

27528 Upon successful completion, `fchmod()` shall return 0. Otherwise, it shall return -1 and set `errno` to
 27529 indicate the error.

27530 **ERRORS**

27531 The `fchmod()` function shall fail if:

27532 [EBADF] The *fildev* argument is not an open file descriptor.

27533 [EPERM] The effective user ID does not match the owner of the file and the process does
 27534 not have appropriate privileges.

27535 [EROFS] The file referred to by *fildev* resides on a read-only file system.

27536 The `fchmod()` function may fail if:

27537 XSI [EINTR] The `fchmod()` function was interrupted by a signal.

27538 XSI [EINVAL] The value of the *mode* argument is invalid.

27539 [EINVAL] The *fildev* argument refers to a pipe and the implementation disallows
 27540 execution of `fchmod()` on a pipe.

27541 **EXAMPLES**27542 **Changing the Current Permissions for a File**

27543 The following example shows how to change the permissions for a file named `/home/cnd/mod1`
 27544 so that the owner and group have read/write/execute permissions, but the world only has
 27545 read/write permissions.

```
27546 #include <sys/stat.h>
27547 #include <fcntl.h>
27548 mode_t mode;
27549 int fildev;
27550 ...
27551 fildev = open("/home/cnd/mod1", O_RDWR);
27552 fchmod(fildev, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```

27553 APPLICATION USAGE

27554 None.

27555 RATIONALE

27556 None.

27557 FUTURE DIRECTIONS

27558 None.

27559 SEE ALSO

27560 *chmod()*, *chown()*, *creat()*, *fcntl()*, *fstatat()*, *fstatofs()*, *mknod()*, *open()*, *read()*, *write()*

27561 XBD <[sys/stat.h](#)>

27562 CHANGE HISTORY

27563 First released in Issue 4, Version 2.

27564 Issue 5

27565 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX
27566 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a
27567 second instance of [EINVAL] is defined in the list of optional errors.

27568 Issue 6

27569 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*
27570 behavior is unspecified for typed memory objects.

27571 **NAME**

27572 fchmodat — change mode of a file relative to directory file descriptor

27573 **SYNOPSIS**

27574 #include <sys/stat.h>

27575 int fchmodat(int *fd*, const char **path*, mode_t *mode*, int *flag*);27576 **DESCRIPTION**27577 Refer to *chmod()*.

27578 **NAME**

27579 fchown — change owner and group of a file

27580 **SYNOPSIS**

27581 #include <unistd.h>

27582 int fchown(int *filde*s, uid_t *owner*, gid_t *group*);27583 **DESCRIPTION**27584 The *fchown()* function shall be equivalent to *chown()* except that the file whose owner and group
27585 are changed is specified by the file descriptor *filde*s.27586 **RETURN VALUE**27587 Upon successful completion, *fchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
27588 indicate the error.27589 **ERRORS**27590 The *fchown()* function shall fail if:27591 [EBADF] The *filde*s argument is not an open file descriptor.27592 [EPERM] The effective user ID does not match the owner of the file or the process does
27593 not have appropriate privileges and `_POSIX_CHOWN_RESTRICTED`
27594 indicates that such privilege is required.27595 [EROFS] The file referred to by *filde*s resides on a read-only file system.27596 The *fchown()* function may fail if:27597 [EINVAL] The owner or group ID is not a value supported by the implementation. The
27598 `OB_XSR` *filde*s argument refers to a pipe or socket or an *fattach()*-ed `STREAM` and the
27599 implementation disallows execution of *fchown()* on a pipe.

27600 [EIO] A physical I/O error has occurred.

27601 [EINTR] The *fchown()* function was interrupted by a signal which was caught.27602 **EXAMPLES**27603 **Changing the Current Owner of a File**27604 The following example shows how to change the owner of a file named `/home/cnd/mod1` to
27605 `“jones”` and the group to `“cnd”`.27606 The numeric value for the user ID is obtained by extracting the user ID from the user database
27607 entry associated with `“jones”`. Similarly, the numeric value for the group ID is obtained by
27608 extracting the group ID from the group database entry associated with `“cnd”`. This example
27609 assumes the calling program has appropriate privileges.

```

27610 #include <sys/types.h>
27611 #include <unistd.h>
27612 #include <fcntl.h>
27613 #include <pwd.h>
27614 #include <grp.h>
27615
27616 struct passwd *pwd;
27617 struct group *grp;
27618 int
27619     filde;
27620
27621 ...
27622 filde = open("/home/cnd/mod1", O_RDWR);
27623 pwd = getpwnam("jones");

```

```
27621     grp = getgrnam("cnd");
27622     fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

27623 APPLICATION USAGE

27624 None.

27625 RATIONALE

27626 None.

27627 FUTURE DIRECTIONS

27628 None.

27629 SEE ALSO

27630 [chown\(\)](#)

27631 XBD [<unistd.h>](#)

27632 CHANGE HISTORY

27633 First released in Issue 4, Version 2.

27634 Issue 5

27635 Moved from X/OPEN UNIX extension to BASE.

27636 Issue 6

27637 The following changes were made to align with the IEEE P1003.1a draft standard:

27638 Clarification is added that a call to *fchown()* may not be allowed on a pipe.

27639 The *fchown()* function is defined as mandatory.

27640 Issue 7

27641 Functionality relating to XSI STREAMS is marked obsolescent.

27642 **NAME**

27643 fchownat — change owner and group of a file relative to directory file descriptor

27644 **SYNOPSIS**

27645 #include <unistd.h>

27646 int fchownat(int *fd*, const char **path*, uid_t *owner*, gid_t *group*,
27647 int *flag*);27648 **DESCRIPTION**27649 Refer to *chown()*.

27650 **NAME**

27651 fclose — close a stream

27652 **SYNOPSIS**

27653 #include <stdio.h>

27654 int fclose(FILE *stream);

27655 **DESCRIPTION**

27656 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27657 conflict between the requirements described here and the ISO C standard is unintentional. This
 27658 volume of POSIX.1-2017 defers to the ISO C standard.

27659 The *fclose()* function shall cause the stream pointed to by *stream* to be flushed and the associated
 27660 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
 27661 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
 27662 disassociated from the file and any buffer set by the *setbuf()* or *setvbuf()* function shall be
 27663 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
 27664 deallocated.

27665 CX If the file is not already at EOF, and the file is one capable of seeking, the file offset of the
 27666 underlying open file description shall be set to the file position of the stream if the stream is the
 27667 active handle to the underlying file description.

27668 The *fclose()* function shall mark for update the last data modification and last file status change
 27669 timestamps of the underlying file, if the stream was writable, and if buffered data remains that
 27670 has not yet been written to the file. The *fclose()* function shall perform the equivalent of a *close()*
 27671 on the file descriptor that is associated with the stream pointed to by *stream*.

27672 After the call to *fclose()*, any use of *stream* results in undefined behavior.

27673 **RETURN VALUE**

27674 CX Upon successful completion, *fclose()* shall return 0; otherwise, it shall return EOF and set *errno*
 27675 to indicate the error.

27676 **ERRORS**

27677 The *fclose()* function shall fail if:

27678 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 27679 the thread would be delayed in the write operation.

27680 CX [EBADF] The file descriptor underlying stream is not valid.

27681 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

27682 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 27683 process.

27684 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 27685 offset maximum associated with the corresponding stream.

27686 CX [EINTR] The *fclose()* function was interrupted by a signal.

27687 CX [EIO] The process is a member of a background process group attempting to write to
 27688 its controlling terminal, TOSTOP is set, the calling thread is not blocking
 27689 SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the
 27690 process is orphaned. This error may also be returned under implementation-
 27691 defined conditions.

- 27692 CX [ENOMEM] The underlying stream was created by *open_memstream()* or
27693 *open_wmemstream()* and insufficient memory is available.
- 27694 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
27695 buffer used by the *fmemopen()* function.
- 27696 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
27697 any process. A SIGPIPE signal shall also be sent to the thread.
- 27698 The *fclose()* function may fail if:
- 27699 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
27700 capabilities of the device.

27701 **EXAMPLES**

27702 None.

27703 **APPLICATION USAGE**

27704 Since after the call to *fclose()* any use of *stream* results in undefined behavior, *fclose()* should not
27705 be used on *stdin*, *stdout*, or *stderr* except immediately before process termination (see XBD
27706 [Section 3.303](#), on page 82), so as to avoid triggering undefined behavior in other standard
27707 interfaces that rely on these streams. If there are any *atexit()* handlers registered by the
27708 application, such a call to *fclose()* should not occur until the last handler is finishing. Once
27709 *fclose()* has been used to close *stdin*, *stdout*, or *stderr*, there is no standard way to reopen any of
27710 these streams.

27711 Use of *freopen()* to change *stdin*, *stdout*, or *stderr* instead of closing them avoids the danger of a
27712 file unexpectedly being opened as one of the special file descriptors `STDIN_FILENO`,
27713 `STDOUT_FILENO`, or `STDERR_FILENO` at a later time in the application.

27714 **RATIONALE**

27715 None.

27716 **FUTURE DIRECTIONS**

27717 None.

27718 **SEE ALSO**

27719 [Section 2.5](#) (on page 495), *atexit()*, *close()*, *fmemopen()*, *fopen()*, *freopen()*, *getrlimit()*,
27720 *open_memstream()*, *ulimit()*

27721 XBD [<stdio.h>](#)

27722 **CHANGE HISTORY**

27723 First released in Issue 1. Derived from Issue 1 of the SVID.

27724 **Issue 5**

27725 Large File Summit extensions are added.

27726 **Issue 6**

27727 Extensions beyond the ISO C standard are marked.

27728 The following new requirements on POSIX implementations derive from alignment with the
27729 Single UNIX Specification:

27730 The [EFBIG] error is added as part of the large file support extensions.

27731 The [ENXIO] optional error condition is added.

27732 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether
27733 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

27734 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/28 is applied, updating the [EAGAIN]
27735 error in the ERRORS section from “the process would be delayed” to “the thread would be
27736 delayed”.

27737 **Issue 7**

27738 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file
27739 descriptors and streams.

27740 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open
27741 Group Technical Standard, 2006, Extended API Set Part 1.

27742 Changes are made related to support for finegrained timestamps.

27743 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0113 [87], XSH/TC1-2008/0114 [79],
27744 and XSH/TC1-2008/0115 [14] are applied.

27745 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0104 [555] is applied.

27746 **NAME**

27747 fcntl — file control

27748 **SYNOPSIS**

27749 #include <fcntl.h>

27750 int fcntl(int *fildev*, int *cmd*, ...);27751 **DESCRIPTION**27752 The *fcntl()* function shall perform the operations described below on open files. The *fildev*
27753 argument is a file descriptor.27754 The available values for *cmd* are defined in <fcntl.h> and are as follows:

27755 **F_DUPFD** Return a new file descriptor which shall be allocated as described in
27756 Section 2.14 (on page 549), except that it shall be the lowest numbered
27757 available file descriptor greater than or equal to the third argument, *arg*,
27758 taken as an integer of type **int**. The new file descriptor shall refer to the
27759 same open file description as the original file descriptor, and shall share
27760 any locks. The FD_CLOEXEC flag associated with the new file descriptor
27761 shall be cleared to keep the file open across calls to one of the *exec*
27762 functions.

27763 **F_DUPFD_CLOEXEC**27764 Like F_DUPFD, but the FD_CLOEXEC flag associated with the new file
27765 descriptor shall be set.27766 **F_GETFD**27767 Get the file descriptor flags defined in <fcntl.h> that are associated with
27768 the file descriptor *fildev*. File descriptor flags are associated with a single
27769 file descriptor and do not affect other file descriptors that refer to the
27770 same file.27770 **F_SETFD**27771 Set the file descriptor flags defined in <fcntl.h>, that are associated with
27772 *fildev*, to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC
27773 flag in the third argument is 0, the file descriptor shall remain open across
27774 the *exec* functions; otherwise, the file descriptor shall be closed upon
27775 successful execution of one of the *exec* functions.27775 **F_GETFL**27776 Get the file status flags and file access modes, defined in <fcntl.h>, for the
27777 file description associated with *fildev*. The file access modes can be
27778 extracted from the return value using the mask O_ACCMODE, which is
27779 defined in <fcntl.h>. File status flags and file access modes are associated
27780 with the file description and do not affect other file descriptors that refer
27781 to the same file with different open file descriptions. The flags returned
27782 may include non-standard file status flags which the application did not
27783 set, provided that these additional flags do not alter the behavior of a
27784 conforming application.27784 **F_SETFL**27785 Set the file status flags, defined in <fcntl.h>, for the file description
27786 associated with *fildev* from the corresponding bits in the third argument,
27787 *arg*, taken as type **int**. Bits corresponding to the file access mode and the
27788 file creation flags, as defined in <fcntl.h>, that are set in *arg* shall be
27789 ignored. If any bits in *arg* other than those mentioned here are changed by
27790 the application, the result is unspecified. If *fildev* does not support non-
27791 blocking operations, it is unspecified whether the O_NONBLOCK flag
27792 will be ignored.

27792	F_GETOWN	If <i>fildev</i> refers to a socket, get the process ID or process group ID specified to receive SIGURG signals when out-of-band data is available. Positive values shall indicate a process ID; negative values, other than -1, shall indicate a process group ID; the value zero shall indicate that no SIGURG signals are to be sent. If <i>fildev</i> does not refer to a socket, the results are unspecified.
27793		
27794		
27795		
27796		
27797		
27798	F_SETOWN	If <i>fildev</i> refers to a socket, set the process ID or process group ID specified to receive SIGURG signals when out-of-band data is available, using the value of the third argument, <i>arg</i> , taken as type int . Positive values shall indicate a process ID; negative values, other than -1, shall indicate a process group ID; the value zero shall indicate that no SIGURG signals are to be sent. Each time a SIGURG signal is sent to the specified process or process group, permission checks equivalent to those performed by <i>kill()</i> shall be performed, as if <i>kill()</i> were called by a process with the same real user ID, effective user ID, and privileges that the process calling <i>fcntl()</i> has at the time of the call; if the <i>kill()</i> call would fail, no signal shall be sent. These permission checks may also be performed by the <i>fcntl()</i> call. If the process specified by <i>arg</i> later terminates, or the process group specified by <i>arg</i> later becomes empty, while still being specified to receive SIGURG signals when out-of-band data is available from <i>fildev</i> , then no signals shall be sent to any subsequently created process that has the same process ID or process group ID, regardless of permission; it is unspecified whether this is achieved by the equivalent of a <i>fcntl(fildev, F_SETOWN, 0)</i> call at the time the process terminates or is waited for or the process group becomes empty, or by other means. If <i>fildev</i> does not refer to a socket, the results are unspecified.
27799		
27800		
27801		
27802		
27803		
27804		
27805		
27806		
27807		
27808		
27809		
27810		
27811		
27812		
27813		
27814		
27815		
27816		
27817		
27818	The following values for <i>cmd</i> are available for advisory record locking. Record locking shall be supported for regular files, and may be supported for other files.	
27819		
27820	F_GETLK	Get any lock which blocks the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type struct flock , defined in <fcntl.h> . The information retrieved shall overwrite the information passed to <i>fcntl()</i> in the structure flock . If no lock is found that would prevent this lock from being created, then the structure shall be left unchanged except for the lock type which shall be set to F_UNLCK.
27821		
27822		
27823		
27824		
27825		
27826	F_SETLK	Set or clear a file segment lock according to the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type struct flock , defined in <fcntl.h> . F_SETLK can establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in <fcntl.h> . If a shared or exclusive lock cannot be set, <i>fcntl()</i> shall return immediately with a return value of -1.
27827		
27828		
27829		
27830		
27831		
27832		
27833	F_SETLKW	This command shall be equivalent to F_SETLK except that if a shared or exclusive lock is blocked by other locks, the thread shall wait until the request can be satisfied. If a signal that is to be caught is received while <i>fcntl()</i> is waiting for a region, <i>fcntl()</i> shall be interrupted. Upon return from the signal handler, <i>fcntl()</i> shall return -1 with <i>errno</i> set to [EINTR], and the lock operation shall not be done.
27834		
27835		
27836		
27837		
27838		
27839	Additional implementation-defined values for <i>cmd</i> may be defined in <fcntl.h> . Their names shall start with F_.	
27840		

27841 When a shared lock is set on a segment of a file, other processes shall be able to set shared locks
 27842 on that segment or a portion of it. A shared lock prevents any other process from setting an
 27843 exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the
 27844 file descriptor was not opened with read access.

27845 An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock
 27846 on any portion of the protected area. A request for an exclusive lock shall fail if the file
 27847 descriptor was not opened with write access.

27848 The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*),
 27849 size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

27850 The value of *l_whence* is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative
 27851 offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file,
 27852 respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of *l_len*
 27853 may be negative (where the definition of **off_t** permits negative values of *l_len*). The *l_pid* field
 27854 is only used with F_GETLK to return the process ID of the process holding a blocking lock. After
 27855 a successful F_GETLK request, when a blocking lock is found, the values returned in the **flock**
 27856 structure shall be as follows:

27857 *l_type* Type of blocking lock found.

27858 *l_whence* SEEK_SET.

27859 *l_start* Start of the blocking lock.

27860 *l_len* Length of the blocking lock.

27861 *l_pid* Process ID of the process that holds the blocking lock.

27862 If the command is F_SETLKW and the process must wait for another process to release a lock,
 27863 then the range of bytes to be locked shall be determined before the *fcntl()* function blocks. If the
 27864 file size or file descriptor seek offset change while *fcntl()* is blocked, this shall not affect the
 27865 range of bytes locked.

27866 If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start+l_len-1*. If *l_len* is
 27867 negative, the area affected shall start at *l_start+l_len* and end at *l_start-1*. Locks may start and
 27868 extend beyond the current end of a file, but shall not extend before the beginning of the file. A
 27869 lock shall be set to extend to the largest possible value of the file offset for that file by setting
 27870 *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to SEEK_SET, the whole file
 27871 shall be locked.

27872 There shall be at most one type of lock set for each byte in the file. Before a successful return
 27873 from an F_SETLK or an F_SETLKW request when the calling process has previously existing
 27874 locks on bytes in the region specified by the request, the previous lock type for each byte in the
 27875 specified region shall be replaced by the new lock type. As specified above under the
 27876 descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request
 27877 (respectively) shall fail or block when another process has existing locks on bytes in the specified
 27878 region and the type of any of those locks conflicts with the type specified in the request.

27879 All locks associated with a file for a given process shall be removed when a file descriptor for
 27880 that file is closed by that process or the process holding that file descriptor terminates. Locks are
 27881 not inherited by a child process.

27882 A potential for deadlock occurs if a process controlling a locked region is put to sleep by
 27883 attempting to lock the locked region of another process. If the system detects that sleeping until
 27884 a locked region is unlocked would cause a deadlock, *fcntl()* shall fail with an [EDEADLK] error.

27885 An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the

27886 requested segment is the maximum value for an object of type **off_t**, when the process has an
 27887 existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall
 27888 be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0.
 27889 Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.

27890 SHM When the file descriptor *fildev* refers to a shared memory object, the behavior of *fcntl()* shall be
 27891 the same as for a regular file except the effect of the following values for the argument *cmd* shall
 27892 be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.

27893 TYM If *fildev* refers to a typed memory object, the result of the *fcntl()* function is unspecified.

27894 RETURN VALUE

27895 Upon successful completion, the value returned shall depend on *cmd* as follows:

27896 F_DUPFD A new file descriptor.

27897 F_DUPFD_CLOEXEC

27898 A new file descriptor.

27899 F_GETFD Value of flags defined in **<fcntl.h>**. The return value shall not be negative.

27900 F_SETFD Value other than -1 .

27901 F_GETFL Value of file status flags and access modes. The return value is not negative.

27902 F_SETFL Value other than -1 .

27903 F_GETLK Value other than -1 .

27904 F_SETLK Value other than -1 .

27905 F_SETLKW Value other than -1 .

27906 F_GETOWN Value of the socket owner process or process group; this will not be -1 .

27907 F_SETOWN Value other than -1 .

27908 Otherwise, -1 shall be returned and *errno* set to indicate the error.

27909 ERRORS

27910 The *fcntl()* function shall fail if:

27911 [EACCES] or [EAGAIN]

27912 The *cmd* argument is F_SETLK; the type of lock (*l_type*) is a shared (F_RDLCK)
 27913 or exclusive (F_WRLCK) lock and the segment of a file to be locked is already
 27914 exclusive-locked by another process, or the type is an exclusive lock and some
 27915 portion of the segment of a file to be locked is already shared-locked or
 27916 exclusive-locked by another process.

27917 [EBADF] The *fildev* argument is not a valid open file descriptor, or the argument *cmd* is
 27918 F_SETLK or F_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK),
 27919 and *fildev* is not a valid file descriptor open for reading, or the type of lock,
 27920 *l_type*, is an exclusive lock (F_WRLCK), and *fildev* is not a valid file descriptor
 27921 open for writing.

27922 [EINTR] The *cmd* argument is F_SETLKW and the function was interrupted by a signal.

27923 [EINVAL] The *cmd* argument is invalid, or the *cmd* argument is F_DUPFD or
 27924 F_DUPFD_CLOEXEC and *arg* is negative or greater than or equal to
 27925 {OPEN_MAX}, or the *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW
 27926 and the data pointed to by *arg* is not valid, or *fildev* refers to a file that does not
 27927 support locking.

27928	[EMFILE]	The argument <i>cmd</i> is F_DUPFD or F_DUPFD_CLOEXEC and all file descriptors available to the process are currently open, or no file descriptors greater than or equal to <i>arg</i> are available.
27929		
27930		
27931	[ENOLCK]	The argument <i>cmd</i> is F_SETLK or F_SETLKW and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
27932		
27933		
27934	[EOVERFLOW]	One of the values to be returned cannot be represented correctly.
27935	[EOVERFLOW]	The <i>cmd</i> argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or, if <i>l_len</i> is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type off_t .
27936		
27937		
27938	[ESRCH]	The <i>cmd</i> argument is F_SETOWN and no process or process group can be found corresponding to that specified by <i>arg</i> .
27939		
27940		The <i>fcntl()</i> function may fail if:
27941	[EDEADLK]	The <i>cmd</i> argument is F_SETLKW, the lock is blocked by a lock from another process, and putting the calling process to sleep to wait for that lock to become free would cause a deadlock.
27942		
27943		
27944	[EINVAL]	The <i>cmd</i> argument is F_SETOWN and the value of the argument is not valid as a process or process group identifier.
27945		
27946	[EPERM]	The <i>cmd</i> argument is F_SETOWN and the calling process does not have permission to send a SIGURG signal to any process specified by <i>arg</i> .
27947		

27948 EXAMPLES

27949 Locking and Unlocking a File

27950 The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then
 27951 later remove it. F_SETLK is used to perform a non-blocking lock request so that the process does
 27952 not have to wait if an incompatible lock is held by another process; instead the process can take
 27953 some other action.

```

27954 #include <stdlib.h>
27955 #include <unistd.h>
27956 #include <fcntl.h>
27957 #include <errno.h>
27958 #include <stdio.h>

27959 int
27960 main(int argc, char *argv[])
27961 {
27962     int fd;
27963     struct flock fl;

27964     fd = open("testfile", O_RDWR);
27965     if (fd == -1)
27966         /* Handle error */;

27967     /* Make a non-blocking request to place a write lock
27968        on bytes 100-109 of testfile */

27969     fl.l_type = F_WRLCK;
27970     fl.l_whence = SEEK_SET;

```

```

27971         fl.l_start = 100;
27972         fl.l_len = 10;

27973         if (fcntl(fd, F_SETLK, &fl) == -1) {
27974             if (errno == EACCES || errno == EAGAIN) {
27975                 printf("Already locked by another process\n");

27976                 /* We cannot get the lock at the moment */

27977             } else {
27978                 /* Handle unexpected error */;
27979             }
27980         } else { /* Lock was granted... */

27981             /* Perform I/O on bytes 100 to 109 of file */

27982             /* Unlock the locked bytes */

27983             fl.l_type = F_UNLCK;
27984             fl.l_whence = SEEK_SET;
27985             fl.l_start = 100;
27986             fl.l_len = 10;
27987             if (fcntl(fd, F_SETLK, &fl) == -1)
27988                 /* Handle error */;
27989         }
27990         exit(EXIT_SUCCESS);
27991     } /* main */

```

27992 Setting the Close-on-Exec Flag

27993 The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```

27994 #include <unistd.h>
27995 #include <fcntl.h>
27996 ...
27997     int flags;

27998     flags = fcntl(fd, F_GETFD);
27999     if (flags == -1)
28000         /* Handle error */;
28001     flags |= FD_CLOEXEC;
28002     if (fcntl(fd, F_SETFD, flags) == -1)
28003         /* Handle error */;

```

28004 APPLICATION USAGE

28005 The *arg* values to `F_GETFD`, `F_SETFD`, `F_GETFL`, and `F_SETFL` all represent flag values to allow
28006 for future growth. Applications using these functions should do a read-modify-write operation
28007 on them, rather than assuming that only the values defined by this volume of POSIX.1-2017 are
28008 valid. It is a common error to forget this, particularly in the case of `F_SETFD`. Some
28009 implementations set additional file status flags to advise the application of default behavior,
28010 even though the application did not request these flags.

28011 On systems which do not perform permission checks at the time of an `fcntl()` call with
28012 `F_SETOWN`, if the permission checks performed at the time the signal is sent disallow sending
28013 the signal to any process, the process that called `fcntl()` has no way of discovering that this has
28014 happened. A call to `kill()` with signal 0 can be used as a prior check of permissions, although this
28015 is no guarantee that permission will be granted at the time a signal is sent, since the target

28016 process(es) could change user IDs or privileges in the meantime.

28017 RATIONALE

28018 The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number
28019 of arguments. It is used because System V uses pointers for the implementation of file locking
28020 functions.

28021 This volume of POSIX.1-2017 permits concurrent read and write access to file data using the
28022 *fcntl()* function; this is a change from the 1984 /usr/group standard and early proposals.
28023 Without concurrency controls, this feature may not be fully utilized without occasional loss of
28024 data.

28025 Data losses occur in several ways. One case occurs when several processes try to update the
28026 same record, without sequencing controls; several updates may occur in parallel and the last
28027 writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing
28028 reorganization. Without exclusive use to the tree segment by the updating process, other reading
28029 processes chance getting lost in the database when the index blocks are split, condensed,
28030 inserted, or deleted. While *fcntl()* is useful for many applications, it is not intended to be overly
28031 general and does not handle the bit-tree example well.

28032 This facility is only required for regular files because it is not appropriate for many devices such
28033 as terminals and network connections.

28034 Since *fcntl()* works with “any file descriptor associated with that file, however it is obtained”,
28035 the file descriptor may have been inherited through a *fork()* or *exec* operation and thus may
28036 affect a file that another process also has open.

28037 The use of the open file description to identify what to lock requires extra calls and presents
28038 problems if several processes are sharing an open file description, but there are too many
28039 implementations of the existing mechanism for this volume of POSIX.1-2017 to use different
28040 specifications.

28041 Another consequence of this model is that closing any file descriptor for a given file (whether or
28042 not it is the same open file description that created the lock) causes the locks on that file to be
28043 relinquished for that process. Equivalently, any close for any file/process pair relinquishes the
28044 locks owned on that file for that process. But note that while an open file description may be
28045 shared through *fork()*, locks are not inherited through *fork()*. Yet locks may be inherited through
28046 one of the *exec* functions.

28047 The identification of a machine in a network environment is outside the scope of this volume of
28048 POSIX.1-2017. Thus, an *l_sysid* member, such as found in System V, is not included in the locking
28049 structure.

28050 Changing of lock types can result in a previously locked region being split into smaller regions.

28051 Mandatory locking was a major feature of the 1984 /usr/group standard.

28052 For advisory file record locking to be effective, all processes that have access to a file must
28053 cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode
28054 record locking is important when it cannot be assumed that all processes are cooperating. For
28055 example, if one user uses an editor to update a file at the same time that a second user executes
28056 another process that updates the same file and if only one of the two processes is using advisory
28057 locking, the processes are not cooperating. Enforcement-mode record locking would protect
28058 against accidental collisions.

28059 Secondly, advisory record locking requires a process using locking to bracket each I/O operation
28060 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a
28061 process can lock the file once and unlock when all I/O operations have been completed.

28062 Enforcement-mode record locking provides a base that can be enhanced; for example, with
 28063 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so
 28064 other processes could read it, but none of them could write it.

28065 Mandatory locks were omitted for several reasons:

- 28066 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most
 28067 implementations; this was confusing, at best.
- 28068 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
- 28069 3. Any publicly readable file could be locked by anyone. Many historical implementations
 28070 keep the password database in a publicly readable file. A malicious user could thus
 28071 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
- 28072 4. Some demand-paged historical implementations offer memory mapped files, and
 28073 enforcement cannot be done on that type of file.

28074 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a
 28075 timeout facility in applications requiring it. This is useful in deadlock detection. Since
 28076 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was
 28077 made optional.

28078 FUTURE DIRECTIONS

28079 None.

28080 SEE ALSO

28081 *alarm()*, *close()*, *exec*, *kill()*, *open()*, *sigaction()*

28082 XBD <fcntl.h>, <signal.h>

28083 CHANGE HISTORY

28084 First released in Issue 1. Derived from Issue 1 of the SVID.

28085 Issue 5

28086 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 28087 Threads Extension.

28088 Large File Summit extensions are added.

28089 Issue 6

28090 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

28091 The following new requirements on POSIX implementations derive from alignment with the
 28092 Single UNIX Specification:

28093 The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
 28094 required for conforming implementations of previous POSIX specifications, it was not
 28095 required for UNIX applications.

28096 In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now
 28097 mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is
 28098 mandated.

28099 In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd*
 28100 is invalid, and two [EOVERFLOW] error conditions are added.

28101 The F_GETOWN and F_SETOWN values are added for sockets.

28102 The following changes were made to align with the IEEE P1003.1a draft standard:

- 28103 Clarification is added that the extent of the bytes locked is determined prior to the
28104 blocking action.
- 28105 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
28106 *fcntl()* results are unspecified for typed memory objects.
- 28107 The normative text is updated to avoid use of the term “must” for application requirements.
- 28108 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/29 is applied, adding the example to the
28109 EXAMPLES section.
- 28110 **Issue 7**
- 28111 Austin Group Interpretation 1003.1-2001 #150 is applied, clarifying the file status flags returned
28112 when *cmd* is F_GETFL.
- 28113 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the
28114 FD_CLOEXEC flag atomically at *open()*, and adding the F_DUPFD_CLOEXEC flag.
- 28115 The optional **<unistd.h>** header is removed from this function, since **<fcntl.h>** now defines
28116 SEEK_SET, SEEK_CUR, and SEEK_END as part of the Base.
- 28117 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0116 [141] is applied.
- 28118 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0105 [835], XSH/TC2-2008/0106 [677],
28119 XSH/TC2-2008/0107 [484], XSH/TC2-2008/0108 [675], and XSH/TC2-2008/0109 [675,677] are
28120 applied.

28121 **NAME**28122 fdatasync — synchronize the data of a file (**REALTIME**)28123 **SYNOPSIS**

```
28124 SIO #include <unistd.h>
28125 int fdatasync(int fildes);
```

28126 **DESCRIPTION**

28127 The *fdatasync()* function shall force all currently queued I/O operations associated with the file
 28128 indicated by file descriptor *fil*des to the synchronized I/O completion state.

28129 The functionality shall be equivalent to *fsync()* with the symbol `_POSIX_SYNCHRONIZED_IO`
 28130 defined, with the exception that all I/O operations shall be completed as defined for
 28131 synchronized I/O data integrity completion.

28132 **RETURN VALUE**

28133 If successful, the *fdatasync()* function shall return the value 0; otherwise, the function shall return
 28134 the value `-1` and set *errno* to indicate the error. If the *fdatasync()* function fails, outstanding I/O
 28135 operations are not guaranteed to have been completed.

28136 **ERRORS**

28137 The *fdatasync()* function shall fail if:

28138 [EBADF] The *fil*des argument is not a valid file descriptor.

28139 [EINVAL] This implementation does not support synchronized I/O for this file.

28140 In the event that any of the queued I/O operations fail, *fdatasync()* shall return the error
 28141 conditions defined for *read()* and *write()*.

28142 **EXAMPLES**

28143 None.

28144 **APPLICATION USAGE**

28145 Note that even if the file descriptor is not open for writing, if there are any pending write
 28146 requests on the underlying file, then that I/O will be completed prior to the return of *fdatasync()*.

28147 **RATIONALE**

28148 None.

28149 **FUTURE DIRECTIONS**

28150 None.

28151 **SEE ALSO**

28152 [aio_fsync\(\)](#), [fcntl\(\)](#), [fsync\(\)](#), [open\(\)](#), [read\(\)](#), [write\(\)](#)

28153 XBD [<unistd.h>](#)

28154 **CHANGE HISTORY**

28155 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

28156 **Issue 6**

28157 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 28158 implementation does not support the Synchronized Input and Output option.

28159 The *fdatasync()* function is marked as part of the Synchronized Input and Output option.



28160 **Issue 7**
28161

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0110 [501] is applied.



28162 **NAME**28163 fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)28164 **SYNOPSIS**

```
28165 OB XSR #include <stropts.h>
28166 int fdetach(const char *path);
```

28167 **DESCRIPTION**

28168 The *fdetach()* function shall detach a STREAMS-based file from the file to which it was attached
 28169 by a previous call to *fattach()*. The *path* argument points to the pathname of the attached
 28170 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A
 28171 successful call to *fdetach()* shall cause all pathnames that named the attached STREAMS file to
 28172 again name the file to which the STREAMS file was attached. All subsequent operations on *path*
 28173 shall operate on the underlying file and not on the STREAMS file.

28174 All open file descriptions established while the STREAMS file was attached to the file referenced
 28175 by *path* shall still refer to the STREAMS file after the *fdetach()* has taken effect.

28176 If there are no open file descriptors or other references to the STREAMS file, then a successful
 28177 call to *fdetach()* shall be equivalent to performing the last *close()* on the attached file.

28178 **RETURN VALUE**

28179 Upon successful completion, *fdetach()* shall return 0; otherwise, it shall return -1 and set *errno* to
 28180 indicate the error.

28181 **ERRORS**

28182 The *fdetach()* function shall fail if:

- 28183 [EACCES] Search permission is denied on a component of the path prefix.
- 28184 [EINVAL] The *path* argument names a file that is not currently attached.
- 28185 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 28186 argument.
- 28187 [ENAMETOOLONG]
 28188 The length of a component of a pathname is longer than {NAME_MAX}.
- 28189 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 28190 [ENOTDIR] A component of the path prefix names an existing file that is neither a
 28191 directory nor a symbolic link to a directory, or the *path* argument contains at
 28192 least one non-*<slash>* character and ends with one or more trailing *<slash>*
 28193 characters and the last pathname component names an existing file that is
 28194 neither a directory nor a symbolic link to a directory.
- 28195 [EPERM] The effective user ID is not the owner of *path* and the process does not have
 28196 appropriate privileges.

28197 The *fdetach()* function may fail if:

- 28198 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 28199 resolution of the *path* argument.
- 28200 [ENAMETOOLONG]
 28201 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 28202 symbolic link produced an intermediate result with a length that exceeds
 28203 {PATH_MAX}.

28204 **EXAMPLES**28205 **Detaching a File**

28206 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to
 28207 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this
 28208 file refer to the underlying file, not to the STREAMS file.

```
28209 #include <stropts.h>
28210 ...
28211     char *pathname = "/tmp/named-STREAM";
28212     int ret;
28213
28214     ret = fdetach(pathname);
```

28214 **APPLICATION USAGE**

28215 None.

28216 **RATIONALE**

28217 None.

28218 **FUTURE DIRECTIONS**

28219 The *fdetach()* function may be removed in a future version.

28220 **SEE ALSO**

28221 *fattach()*

28222 XBD [<stropts.h>](#)

28223 **CHANGE HISTORY**

28224 First released in Issue 4, Version 2.

28225 **Issue 5**

28226 Moved from X/OPEN UNIX extension to BASE.

28227 **Issue 6**

28228 The normative text is updated to avoid use of the term “must” for application requirements.

28229 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 28230 [ELOOP] error condition is added.

28231 **Issue 7**

28232 Austin Group Interpretation 1003.1-2001 #143 is applied.

28233 The *fdetach()* function is marked obsolescent.

28234 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
 28235 pathname exists but is not a directory or a symbolic link to a directory.

28236 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0117 [324] and XSH/TC1-2008/0118
 28237 [291] are applied.

28238 **NAME**

28239 fdim, fdimf, fdiml — compute positive difference between two floating-point numbers

28240 **SYNOPSIS**

```
28241 #include <math.h>
28242 double fdim(double x, double y);
28243 float fdimf(float x, float y);
28244 long double fdiml(long double x, long double y);
```

28245 **DESCRIPTION**

28246 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28247 conflict between the requirements described here and the ISO C standard is unintentional. This
 28248 volume of POSIX.1-2017 defers to the ISO C standard.

28249 These functions shall determine the positive difference between their arguments. If x is greater
 28250 than y , $x-y$ is returned. If x is less than or equal to y , +0 is returned.

28251 An application wishing to check for error situations should set *errno* to zero and call
 28252 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 28253 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 28254 zero, an error has occurred.

28255 **RETURN VALUE**

28256 Upon successful completion, these functions shall return the positive difference value.

28257 If $x-y$ is positive and overflows, a range error shall occur and *fdim()*, *fdimf()*, and *fdiml()* shall
 28258 return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

28259 If the correct value would cause underflow, a range error may occur, and *fdim()*, *fdimf()*, and
 28260 MXX *fdiml()* shall return the correct value, or (if the IEC 60559 Floating-Point option is not
 28261 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 28262 FLT_MIN, and LDBL_MIN, respectively.

28263 MX If x or y is NaN, a NaN shall be returned.

28264 **ERRORS**28265 The *fdim()* function shall fail if:

28266 Range Error The result overflows.

28267 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28268 then *errno* shall be set to [ERANGE]. If the integer expression
 28269 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 28270 floating-point exception shall be raised.

28271 The *fdim()* function may fail if:

28272 Range Error The result underflows.

28273 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28274 then *errno* shall be set to [ERANGE]. If the integer expression
 28275 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 28276 floating-point exception shall be raised.

28277 **EXAMPLES**

28278 None.

28279 **APPLICATION USAGE**

28280 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
28281 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

28282 **RATIONALE**

28283 None.

28284 **FUTURE DIRECTIONS**

28285 None.

28286 **SEE ALSO**28287 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [fmax\(\)](#), [fmin\(\)](#)28288 [Section 4.20](#) (on page 117), [<math.h>](#)28289 **CHANGE HISTORY**

28290 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

28291 **Issue 7**

28292 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0119 [68,428] and
28293 XSH/TC1-2008/0120 [68,428] are applied.

28294 **NAME**

28295 fdopen — associate a stream with a file descriptor

28296 **SYNOPSIS**

```
28297 CX #include <stdio.h>
28298 FILE *fdopen(int fil-des, const char *mode);
```

28299 **DESCRIPTION**28300 The *fdopen()* function shall associate a stream with a file descriptor.28301 The *mode* argument is a character string having one of the following values:

28302	<i>r</i> or <i>rb</i>	Open a file for reading.
28303	<i>w</i> or <i>wb</i>	Open a file for writing.
28304	<i>a</i> or <i>ab</i>	Open a file for writing at end-of-file.
28305	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open a file for update (reading and writing).
28306	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Open a file for update (reading and writing).
28307	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Open a file for update (reading and writing) at end-of-file.

28308 The meaning of these flags is exactly as specified in *fopen()*, except that modes beginning with *w*
28309 shall not cause truncation of the file.28310 Additional values for the *mode* argument may be supported by an implementation.28311 The application shall ensure that the mode of the stream as expressed by the *mode* argument is
28312 allowed by the file access mode of the open file description to which *fil-des* refers. The file
28313 position indicator associated with the new stream is set to the position indicated by the file offset
28314 associated with the file descriptor.28315 The error and end-of-file indicators for the stream shall be cleared. The *fdopen()* function may
28316 cause the last data access timestamp of the underlying file to be marked for update.28317 SHM If *fil-des* refers to a shared memory object, the result of the *fdopen()* function is unspecified.28318 TYM If *fil-des* refers to a typed memory object, the result of the *fdopen()* function is unspecified.28319 The *fdopen()* function shall preserve the offset maximum previously set for the open file
28320 description corresponding to *fil-des*.28321 **RETURN VALUE**28322 Upon successful completion, *fdopen()* shall return a pointer to a stream; otherwise, a null pointer
28323 shall be returned and *errno* set to indicate the error.28324 **ERRORS**28325 The *fdopen()* function shall fail if:

28326 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

28327 The *fdopen()* function may fail if:28328 [EBADF] The *fil-des* argument is not a valid file descriptor.28329 [EINVAL] The *mode* argument is not a valid mode.

28330 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

28331 [ENOMEM] Insufficient space to allocate a buffer.

28332 **EXAMPLES**

28333 None.

28334 **APPLICATION USAGE**

28335 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but
28336 do not return streams.

28337 **RATIONALE**

28338 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, *fcntl()*, or *socket()*;
28339 inherited through *fork()*, *posix_spawn()*, or *exec*; or perhaps obtained by other means.

28340 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write
28341 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument
28342 formats that include a *b* are allowed for consistency with the ISO C standard function *fopen()*.
28343 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of
28344 POSIX.1-2017, a good implementation of append (*a*) mode would cause the *O_APPEND* flag to
28345 be set.

28346 **FUTURE DIRECTIONS**

28347 None.

28348 **SEE ALSO**

28349 [Section 2.5.1](#) (on page 497), [fclose\(\)](#), [fmemopen\(\)](#), [fopen\(\)](#), [open\(\)](#), [open_memstream\(\)](#),
28350 [posix_spawn\(\)](#), [socket\(\)](#)

28351 XBD [<stdio.h>](#)

28352 **CHANGE HISTORY**

28353 First released in Issue 1. Derived from Issue 1 of the SVID.

28354 **Issue 5**

28355 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

28356 Large File Summit extensions are added.

28357 **Issue 6**

28358 The following new requirements on POSIX implementations derive from alignment with the
28359 Single UNIX Specification:

28360 In the DESCRIPTION, the use and setting of the *mode* argument are changed to include
28361 binary streams.

28362 In the DESCRIPTION, text is added for large file support to indicate setting of the offset
28363 maximum in the open file description.

28364 All errors identified in the ERRORS section are added.

28365 In the DESCRIPTION, text is added that the *fdopen()* function may cause *st_atime* to be
28366 updated.

28367 The following changes were made to align with the IEEE P1003.1a draft standard:

28368 Clarification is added that it is the responsibility of the application to ensure that the mode
28369 is compatible with the open file descriptor.

28370 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
28371 *fdopen()* results are unspecified for typed memory objects.

- 28372 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/30 is applied, making corrections to the
- 28373 RATIONALE.
- 28374 **Issue 7**
- 28375 SD5-XSH-ERN-149 is applied, adding the {STREAM_MAX} [EMFILE] error condition.
- 28376 Changes are made related to support for finegrained timestamps.
- 28377 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0121 [409] is applied.

28378 **NAME**

28379 fdopendir, opendir — open directory associated with file descriptor

28380 **SYNOPSIS**

```
28381 #include <dirent.h>
28382 DIR *fdopendir(int fd);
28383 DIR *opendir(const char *dirname);
```

28384 **DESCRIPTION**

28385 The *fdopendir()* function shall be equivalent to the *opendir()* function except that the directory is
 28386 specified by a file descriptor rather than by a name. The file offset associated with the file
 28387 descriptor at the time of the call determines which entries are returned.

28388 Upon successful return from *fdopendir()*, the file descriptor is under the control of the system,
 28389 and if any attempt is made to close the file descriptor, or to modify the state of the associated
 28390 XSI description, other than by means of *closedir()*, *readdir()*, *readdir_r()*, *rewinddir()*, or *seekdir()*, the
 28391 behavior is undefined. Upon calling *closedir()* the file descriptor shall be closed.

28392 It is unspecified whether the FD_CLOEXEC flag will be set on the file descriptor by a successful
 28393 call to *fdopendir()*.

28394 The *opendir()* function shall open a directory stream corresponding to the directory named by
 28395 the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is
 28396 implemented using a file descriptor, applications shall only be able to open up to a total of
 28397 {OPEN_MAX} files and directories.

28398 If the type **DIR** is implemented using a file descriptor, the descriptor shall be obtained as if the
 28399 O_DIRECTORY flag was passed to *open()*.

28400 **RETURN VALUE**

28401 Upon successful completion, these functions shall return a pointer to an object of type **DIR**.
 28402 Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

28403 **ERRORS**

28404 The *fdopendir()* function shall fail if:

28405 [EBADF] The *fd* argument is not a valid file descriptor open for reading.

28406 [ENOTDIR] The descriptor *fd* is not associated with a directory.

28407 The *opendir()* function shall fail if:

28408 [EACCES] Search permission is denied for the component of the path prefix of *dirname* or
 28409 read permission is denied for *dirname*.

28410 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname*
 28411 argument.

28412 [ENAMETOOLONG]
 28413 The length of a component of a pathname is longer than {NAME_MAX}.

28414 [ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an
 28415 empty string.

28416 [ENOTDIR] A component of *dirname* names an existing file that is neither a directory nor a
 28417 symbolic link to a directory.

- 28418 The *opendir()* function may fail if:
- 28419 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
28420 resolution of the *dirname* argument.
- 28421 [EMFILE] All file descriptors available to the process are currently open.
- 28422 [ENAMETOOLONG]
28423 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
28424 symbolic link produced an intermediate result with a length that exceeds
28425 {PATH_MAX}.
- 28426 [ENFILE] Too many files are currently open in the system.

28427 EXAMPLES

28428 Open a Directory Stream

28429 The following program fragment demonstrates how the *opendir()* function is used.

```
28430 #include <dirent.h>
28431 ...
28432     DIR *dir;
28433     struct dirent *dp;
28434 ...
28435     if ((dir = opendir(".")) == NULL) {
28436         perror("Cannot open .");
28437         exit(1);
28438     }
28439     while ((dp = readdir(dir)) != NULL) {
28440     ...
```

28441 Find And Open a File

28442 The following program searches through a given directory looking for files whose name does
28443 not begin with a dot and whose size is larger than 1 MiB.

```
28444 #include <stdio.h>
28445 #include <dirent.h>
28446 #include <fcntl.h>
28447 #include <sys/stat.h>
28448 #include <stdint.h>
28449 #include <stdlib.h>
28450 #include <unistd.h>
28451 int
28452 main(int argc, char *argv[])
28453 {
28454     struct stat statbuf;
28455     DIR *d;
28456     struct dirent *dp;
28457     int dfd, ffd;
28458     if ((d = fdopendir((dfd = open("./tmp", O_RDONLY)))) == NULL) {
28459         fprintf(stderr, "Cannot open ./tmp directory\n");
28460         exit(1);
```

```

28461     }
28462     while ((dp = readdir(d)) != NULL) {
28463         if (dp->d_name[0] == '.')
28464             continue;
28465         /* there is a possible race condition here as the file
28466          * could be renamed between the readdir and the open */
28467         if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
28468             perror(dp->d_name);
28469             continue;
28470         }
28471         if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {
28472             /* found it ... */
28473             printf("%s: %jdK\n", dp->d_name,
28474                 (intmax_t)(statbuf.st_size / 1024));
28475         }
28476         close(ffd);
28477     }
28478     closedir(d); // note this implicitly closes dfd
28479     return 0;
28480 }

```

APPLICATION USAGE

The `opendir()` function should be used in conjunction with `readdir()`, `closedir()`, and `rewinddir()` to examine the contents of the directory (see the EXAMPLES section in `readdir()`). This method is recommended for portability.

RATIONALE

The purpose of the `fdopendir()` function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to `opendir()`, resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-2017 does not mandate that the directory stream be implemented using file descriptors. The description of `closedir()` clarifies that if a file descriptor is used for the directory stream, it is mandatory that `closedir()` deallocate the file descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the `FD_CLOEXEC` had been set for the file descriptor.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2017 does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a “normal” entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-2017 imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a `dirp` value or a `dirent` structure value after a directory stream has been closed or after a `fork()` or one of the `exec` function calls.

28508 FUTURE DIRECTIONS

28509 None.

28510 SEE ALSO

28511 *closedir()*, *dirfd()*, *fstatat()*, *open()*, *readdir()*, *rewinddir()*, *symlink()*

28512 XBD **<dirent.h>**, **<sys/types.h>**

28513 CHANGE HISTORY

28514 First released in Issue 2.

28515 Issue 6

28516 In the SYNOPSIS, the optional include of the **<sys/types.h>** header is removed.

28517 The following new requirements on POSIX implementations derive from alignment with the
28518 Single UNIX Specification:

28519 The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
28520 required for conforming implementations of previous POSIX specifications, it was not
28521 required for UNIX applications.

28522 The [ELOOP] mandatory error condition is added.

28523 A second [ENAMETOOLONG] is added as an optional error condition.

28524 The following changes were made to align with the IEEE P1003.1a draft standard:

28525 The [ELOOP] optional error condition is added.

28526 Issue 7

28527 Austin Group Interpretation 1003.1-2001 #143 is applied.

28528 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

28529 The *fdopendir()* function is added from The Open Group Technical Standard, 2006, Extended API
28530 Set Part 2.

28531 An additional example is added.

28532 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0122 [422] and XSH/TC1-2008/0123
28533 [324] are applied.

28534 **NAME**

28535 feclearexcept — clear floating-point exception

28536 **SYNOPSIS**

28537 #include <fenv.h>

28538 int feclearexcept(int *excepts*);

28539 **DESCRIPTION**

28540 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
28541 conflict between the requirements described here and the ISO C standard is unintentional. This
28542 volume of POSIX.1-2017 defers to the ISO C standard.

28543 The *feclearexcept()* function shall attempt to clear the supported floating-point exceptions
28544 represented by *excepts*.

28545 **RETURN VALUE**

28546 If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept()*
28547 shall return zero. Otherwise, it shall return a non-zero value.

28548 **ERRORS**

28549 No errors are defined.

28550 **EXAMPLES**

28551 None.

28552 **APPLICATION USAGE**

28553 None.

28554 **RATIONALE**

28555 None.

28556 **FUTURE DIRECTIONS**

28557 None.

28558 **SEE ALSO**

28559 [*fegetexceptflag\(\)*](#), [*feraiseexcept\(\)*](#), [*fetestexcept\(\)*](#)

28560 XBD [**<fenv.h>**](#)

28561 **CHANGE HISTORY**

28562 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28563 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

28564 **NAME**

28565 fegetenv, fesetenv — get and set current floating-point environment

28566 **SYNOPSIS**

```
28567 #include <fenv.h>
28568 int fegetenv(fenv_t *envp);
28569 int fesetenv(const fenv_t *envp);
```

28570 **DESCRIPTION**

28571 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28572 conflict between the requirements described here and the ISO C standard is unintentional. This
28573 volume of POSIX.1-2017 defers to the ISO C standard.

28574 The *fegetenv()* function shall attempt to store the current floating-point environment in the object
28575 pointed to by *envp*.

28576 The *fesetenv()* function shall attempt to establish the floating-point environment represented by
28577 the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to
28578 *fegetenv()* or *fehldexcept()*, or equal a floating-point environment macro. The *fesetenv()* function
28579 does not raise floating-point exceptions, but only installs the state of the floating-point status
28580 flags represented through its argument.

28581 **RETURN VALUE**

28582 If the representation was successfully stored, *fegetenv()* shall return zero. Otherwise, it shall
28583 return a non-zero value. If the environment was successfully established, *fesetenv()* shall return
28584 zero. Otherwise, it shall return a non-zero value.

28585 **ERRORS**

28586 No errors are defined.

28587 **EXAMPLES**

28588 None.

28589 **APPLICATION USAGE**

28590 None.

28591 **RATIONALE**

28592 None.

28593 **FUTURE DIRECTIONS**

28594 None.

28595 **SEE ALSO**28596 *fehldexcept()*, *feupdateenv()*28597 XBD [<fenv.h>](#)28598 **CHANGE HISTORY**

28599 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28600 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

28601 **NAME**

28602 fegetexceptflag, fesetexceptflag ‡get and set floating-point status flags

28603 **SYNOPSIS**

28604 #include <fenv.h>

28605 int fegetexceptflag(fexcept_t *flagp, int excepts);

28606 int fesetexceptflag(const fexcept_t *flagp, int excepts);

28607 **DESCRIPTION**

28608 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28609 conflict between the requirements described here and the ISO C standard is unintentional. This
 28610 volume of POSIX.1-2017 defers to the ISO C standard.

28611 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of
 28612 the states of the floating-point status flags indicated by the argument *excepts* in the object
 28613 pointed to by the argument *flagp*.

28614 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the
 28615 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by
 28616 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument
 28617 represented at least those floating-point exceptions represented by the argument *excepts*. This
 28618 function does not raise floating-point exceptions, but only sets the state of the flags.

28619 **RETURN VALUE**

28620 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it
 28621 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions
 28622 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero
 28623 value.

28624 **ERRORS**

28625 No errors are defined.

28626 **EXAMPLES**

28627 None.

28628 **APPLICATION USAGE**

28629 None.

28630 **RATIONALE**

28631 None.

28632 **FUTURE DIRECTIONS**

28633 None.

28634 **SEE ALSO**28635 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*

28636 XBD <fenv.h>

28637 **CHANGE HISTORY**

28638 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28639 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

28640 **NAME**

28641 fegetround, fesetround — get and set current rounding direction

28642 **SYNOPSIS**

```
28643 #include <fenv.h>
28644 int fegetround(void);
28645 int fesetround(int round);
```

28646 **DESCRIPTION**

28647 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28648 conflict between the requirements described here and the ISO C standard is unintentional. This
 28649 volume of POSIX.1-2017 defers to the ISO C standard.

28650 The *fegetround()* function shall get the current rounding direction.

28651 The *fesetround()* function shall establish the rounding direction represented by its argument
 28652 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding
 28653 direction is not changed.

28654 **RETURN VALUE**

28655 The *fegetround()* function shall return the value of the rounding direction macro representing the
 28656 current rounding direction or a negative value if there is no such rounding direction macro or
 28657 the current rounding direction is not determinable.

28658 The *fesetround()* function shall return a zero value if and only if the requested rounding direction
 28659 was established.

28660 **ERRORS**

28661 No errors are defined.

28662 **EXAMPLES**

28663 The following example saves, sets, and restores the rounding direction, reporting an error and
 28664 aborting if setting the rounding direction fails:

```
28665 #include <fenv.h>
28666 #include <assert.h>
28667 void f(int round_dir)
28668 {
28669     #pragma STDC FENV_ACCESS ON
28670     int save_round;
28671     int setround_ok;
28672     save_round = fegetround();
28673     setround_ok = fesetround(round_dir);
28674     assert(setround_ok == 0);
28675     /* ... */
28676     fesetround(save_round);
28677     /* ... */
28678 }
```

28679 **APPLICATION USAGE**

28680 None.

28681 **RATIONALE**

28682 None.

28683 **FUTURE DIRECTIONS**

28684 None.

28685 **SEE ALSO**

28686 XBD [<fenv.h>](#)

28687 **CHANGE HISTORY**

28688 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28689 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

28690 **NAME**

28691 feholdexcept — save current floating-point environment

28692 **SYNOPSIS**

28693 #include <fenv.h>

28694 int feholdexcept(fenv_t *envp);

28695 **DESCRIPTION**

28696 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28697 conflict between the requirements described here and the ISO C standard is unintentional. This
28698 volume of POSIX.1-2017 defers to the ISO C standard.

28699 The *feholdexcept()* function shall save the current floating-point environment in the object
28700 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
28701 floating-point exceptions) mode, if available, for all floating-point exceptions.

28702 **RETURN VALUE**

28703 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception
28704 handling was successfully installed.

28705 **ERRORS**

28706 No errors are defined.

28707 **EXAMPLES**

28708 None.

28709 **APPLICATION USAGE**

28710 None.

28711 **RATIONALE**

28712 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard
28713 implementations which have the default non-stop mode and at least one other mode for trap
28714 handling or aborting. If the implementation provides only the non-stop mode, then installing the
28715 non-stop mode is trivial.

28716 **FUTURE DIRECTIONS**

28717 None.

28718 **SEE ALSO**28719 *fegetenv()*, *feupdateenv()*28720 XBD <[fenv.h](#)>28721 **CHANGE HISTORY**

28722 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28723 **NAME**

28724 feof — test end-of-file indicator on a stream

28725 **SYNOPSIS**

28726 #include <stdio.h>

28727 int feof(FILE *stream);

28728 **DESCRIPTION**

28729 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28730 conflict between the requirements described here and the ISO C standard is unintentional. This
28731 volume of POSIX.1-2017 defers to the ISO C standard.

28732 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.28733 CX The *feof()* function shall not change the setting of *errno* if *stream* is valid.28734 **RETURN VALUE**28735 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.28736 **ERRORS**

28737 No errors are defined.

28738 **EXAMPLES**

28739 None.

28740 **APPLICATION USAGE**

28741 None.

28742 **RATIONALE**

28743 None.

28744 **FUTURE DIRECTIONS**

28745 None.

28746 **SEE ALSO**28747 *clearerr()*, *ferror()*, *fopen()*

28748 XBD <stdio.h>

28749 **CHANGE HISTORY**

28750 First released in Issue 1. Derived from Issue 1 of the SVID.

28751 **Issue 7**

28752 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0124 [401] is applied.

28753 **NAME**28754 `feraiseexcept` \ddagger raise floating-point exception28755 **SYNOPSIS**28756 `#include <fenv.h>`28757 `int feraiseexcept(int excepts);`28758 **DESCRIPTION**

28759 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28760 conflict between the requirements described here and the ISO C standard is unintentional. This
 28761 volume of POSIX.1-2017 defers to the ISO C standard.

28762 The `feraiseexcept()` function shall attempt to raise the supported floating-point exceptions
 28763 represented by the `excepts` argument. The order in which these floating-point exceptions are
 28764 MX raised is unspecified, except that if the `excepts` argument represents IEC 60559 valid coincident
 28765 floating-point exceptions for atomic operations (namely overflow and inexact, or underflow and
 28766 inexact), then overflow or underflow shall be raised before inexact. Whether the `feraiseexcept()`
 28767 function additionally raises the inexact floating-point exception whenever it raises the overflow
 28768 or underflow floating-point exception is implementation-defined.

28769 **RETURN VALUE**

28770 If the argument is zero or if all the specified exceptions were successfully raised, `feraiseexcept()`
 28771 shall return zero. Otherwise, it shall return a non-zero value.

28772 **ERRORS**

28773 No errors are defined.

28774 **EXAMPLES**

28775 None.

28776 **APPLICATION USAGE**

28777 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic
 28778 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

28779 **RATIONALE**

28780 Raising overflow or underflow is allowed to also raise inexact because on some architectures the
 28781 only practical way to raise an exception is to execute an instruction that has the exception as a
 28782 side-effect. The function is not restricted to accept only valid coincident expressions for atomic
 28783 operations, so the function can be used to raise exceptions accrued over several operations.

28784 **FUTURE DIRECTIONS**

28785 None.

28786 **SEE ALSO**28787 [`feclearexcept\(\)`](#), [`fetestexceptflag\(\)`](#), [`fetestexcept\(\)`](#)28788 XBD [`<fenv.h>`](#)28789 **CHANGE HISTORY**

28790 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28791 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

28792 **Issue 7**

28793 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0111 [543] is applied.

28794 **NAME**28795 `ferror` — test error indicator on a stream28796 **SYNOPSIS**28797 `#include <stdio.h>`28798 `int ferror(FILE *stream);`28799 **DESCRIPTION**

28800 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28801 conflict between the requirements described here and the ISO C standard is unintentional. This
28802 volume of POSIX.1-2017 defers to the ISO C standard.

28803 The `ferror()` function shall test the error indicator for the stream pointed to by `stream`.

28804 CX The `ferror()` function shall not change the setting of `errno` if `stream` is valid.

28805 **RETURN VALUE**

28806 The `ferror()` function shall return non-zero if and only if the error indicator is set for `stream`.

28807 **ERRORS**

28808 No errors are defined.

28809 **EXAMPLES**

28810 None.

28811 **APPLICATION USAGE**

28812 None.

28813 **RATIONALE**

28814 None.

28815 **FUTURE DIRECTIONS**

28816 None.

28817 **SEE ALSO**

28818 [*clearerr\(\)*](#), [*feof\(\)*](#), [*fopen\(\)*](#)

28819 XBD [**<stdio.h>**](#)

28820 **CHANGE HISTORY**

28821 First released in Issue 1. Derived from Issue 1 of the SVID.

28822 **Issue 7**

28823 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0125 [401] is applied.

28824 **NAME**

28825 fesetenv — set current floating-point environment

28826 **SYNOPSIS**

28827 #include <fenv.h>

28828 int fesetenv(const fenv_t *envp);

28829 **DESCRIPTION**28830 Refer to *fegetenv()*.

28831 **NAME**

28832 fesetexceptflag ‡'set floating-point status flags

28833 **SYNOPSIS**

28834 #include <fenv.h>

28835 int fesetexceptflag(const fexcept_t *flagp, int excepts);

28836 **DESCRIPTION**

28837 Refer to *fegetexceptflag()*.

28838 **NAME**

28839 fesetround — set current rounding direction

28840 **SYNOPSIS**

28841 #include <fenv.h>

28842 int fesetround(int *round*);

28843 **DESCRIPTION**

28844 Refer to *fegetround()*.

28845 **NAME**

28846 fetestexcept ‡test floating-point exception flags

28847 **SYNOPSIS**

28848 #include <fenv.h>

28849 int fetestexcept(int *excepts*);28850 **DESCRIPTION**

28851 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28852 conflict between the requirements described here and the ISO C standard is unintentional. This
 28853 volume of POSIX.1-2017 defers to the ISO C standard.

28854 The *fetestexcept()* function shall determine which of a specified subset of the floating-point
 28855 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to
 28856 be queried.

28857 **RETURN VALUE**

28858 The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point
 28859 exception macros corresponding to the currently set floating-point exceptions included in
 28860 *excepts*.

28861 **ERRORS**

28862 No errors are defined.

28863 **EXAMPLES**

28864 The following example calls function *f()* if an invalid exception is set, and then function *g()* if an
 28865 overflow exception is set:

```
28866       #include <fenv.h>
28867       /* ... */
28868       {
28869           #pragma STDC FENV_ACCESS ON
28870           int set_excepts;
28871           feclearexcept(FE_INVALID | FE_OVERFLOW);
28872           // maybe raise exceptions
28873           set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
28874           if (set_excepts & FE_INVALID) f();
28875           if (set_excepts & FE_OVERFLOW) g();
28876           /* ... */
28877       }
```

28878 **APPLICATION USAGE**

28879 None.

28880 **RATIONALE**

28881 None.

28882 **FUTURE DIRECTIONS**

28883 None.

28884 **SEE ALSO**28885 *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*28886 XBD <[fenv.h](#)>

28887 **CHANGE HISTORY**

28888 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28889 **NAME**

28890 feupdateenv — update floating-point environment

28891 **SYNOPSIS**

```
28892 #include <fenv.h>
28893 int feupdateenv(const fenv_t *envp);
```

28894 **DESCRIPTION**

28895 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28896 conflict between the requirements described here and the ISO C standard is unintentional. This
 28897 volume of POSIX.1-2017 defers to the ISO C standard.

28898 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in
 28899 its automatic storage, attempt to install the floating-point environment represented by the object
 28900 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument
 28901 *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point
 28902 environment macro.

28903 **RETURN VALUE**

28904 The *feupdateenv()* function shall return a zero value if and only if all the required actions were
 28905 successfully carried out.

28906 **ERRORS**

28907 No errors are defined.

28908 **EXAMPLES**

28909 The following example shows sample code to hide spurious underflow floating-point
 28910 exceptions:

```
28911 #include <fenv.h>
28912 double f(double x)
28913 {
28914     #pragma STDC FENV_ACCESS ON
28915     double result;
28916     fenv_t save_env;
28917     feholdexcept(&save_env);
28918     // compute result
28919     if (/* test spurious underflow */)
28920         feclearexcept(FE_UNDERFLOW);
28921     feupdateenv(&save_env);
28922     return result;
28923 }
```

28924 **APPLICATION USAGE**

28925 None.

28926 **RATIONALE**

28927 None.

28928 **FUTURE DIRECTIONS**

28929 None.

28930 **SEE ALSO**28931 *fegetenv()*, *feholdexcept()*28932 XBD [<fenv.h>](#)

28933 **CHANGE HISTORY**

- 28934 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.
- 28935 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

28936 **NAME**

28937 fexecve ‡'execute a file

28938 **SYNOPSIS**

28939 #include <unistd.h>

28940 int fexecve(int *fd*, char *const *argv*[], char *const *envp*[]);28941 **DESCRIPTION**28942 Refer to *exec*.

28943 **NAME**

28944 fflush — flush a stream

28945 **SYNOPSIS**

28946 #include <stdio.h>

28947 int fflush(FILE *stream);

28948 **DESCRIPTION**

28949 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28950 conflict between the requirements described here and the ISO C standard is unintentional. This
 28951 volume of POSIX.1-2017 defers to the ISO C standard.

28952 If *stream* points to an output stream or an update stream in which the most recent operation was
 28953 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the
 28954 last data modification and last file status change timestamps of the underlying file shall be
 28955 marked for update.

28956 For a stream open for reading with an underlying file description, if the file is not already at
 28957 EOF, and the file is one capable of seeking, the file offset of the underlying open file description
 28958 shall be set to the file position of the stream, and any characters pushed back onto the stream by
 28959 *ungetc()* or *ungetwc()* that have not subsequently been read from the stream shall be discarded
 28960 (without further changing the file offset).

28961 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the
 28962 behavior is defined above.

28963 **RETURN VALUE**

28964 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for
 28965 CX the stream, return EOF, and set *errno* to indicate the error.

28966 **ERRORS**28967 The *fflush()* function shall fail if:

28968 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 28969 the thread would be delayed in the write operation.

28970 CX [EBADF] The file descriptor underlying *stream* is not valid.

28971 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

28972 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 28973 process.

28974 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 28975 offset maximum associated with the corresponding stream.

28976 CX [EINTR] The *fflush()* function was interrupted by a signal.

28977 CX [EIO] The process is a member of a background process group attempting to write to
 28978 its controlling terminal, TOSTOP is set, the calling thread is not blocking
 28979 SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the
 28980 process is orphaned. This error may also be returned under implementation-
 28981 defined conditions.

28982 CX [ENOMEM] The underlying stream was created by *open_memstream()* or
 28983 *open_wmemstream()* and insufficient memory is available.

28984 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
 28985 buffer used by the *fmemopen()* function.

28986 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
 28987 any process. A SIGPIPE signal shall also be sent to the thread.

28988 The *fflush()* function may fail if:

28989 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 28990 capabilities of the device.

28991 EXAMPLES

28992 Sending Prompts to Standard Output

28993 The following example uses *printf()* calls to print a series of prompts for information the user
 28994 must enter from standard input. The *fflush()* calls force the output to standard output. The
 28995 *fflush()* function is used because standard output is usually buffered and the prompt may not
 28996 immediately be printed on the output or terminal. The *getline()* function calls read strings from
 28997 standard input and place the results in variables, for use later in the program.

```

28998 char *user;
28999 char *oldpasswd;
29000 char *newpasswd;
29001 ssize_t llen;
29002 size_t blen;
29003 struct termios term;
29004 tcflag_t saveflag;

29005 printf("User name: ");
29006 fflush(stdout);
29007 blen = 0;
29008 llen = getline(&user, &blen, stdin);
29009 user[llen-1] = 0;
29010 tcgetattr(fileno(stdin), &term);
29011 saveflag = term.c_lflag;
29012 term.c_lflag &= ~ECHO;
29013 tcsetattr(fileno(stdin), TCSANOW, &term);
29014 printf("Old password: ");
29015 fflush(stdout);
29016 blen = 0;
29017 llen = getline(&oldpasswd, &blen, stdin);
29018 oldpasswd[llen-1] = 0;

29019 printf("\nNew password: ");
29020 fflush(stdout);
29021 blen = 0;
29022 llen = getline(&newpasswd, &blen, stdin);
29023 newpasswd[llen-1] = 0;
29024 term.c_lflag = saveflag;
29025 tcsetattr(fileno(stdin), TCSANOW, &term);
29026 free(user);
29027 free(oldpasswd);
29028 free(newpasswd);

```

29029 APPLICATION USAGE

29030 None.

29031 RATIONALE

29032 Data buffered by the system may make determining the validity of the position of the current
29033 file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()*
29034 on streams open for *read()* is not mandated by POSIX.1-2017.

29035 FUTURE DIRECTIONS

29036 None.

29037 SEE ALSO

29038 [Section 2.5](#) (on page 495), *fmemopen()*, *getrlimit()*, *open_memstream()*, *ulimit()*

29039 XBD [<stdio.h>](#)

29040 CHANGE HISTORY

29041 First released in Issue 1. Derived from Issue 1 of the SVID.

29042 Issue 5

29043 Large File Summit extensions are added.

29044 Issue 6

29045 Extensions beyond the ISO C standard are marked.

29046 The following new requirements on POSIX implementations derive from alignment with the
29047 Single UNIX Specification:

29048 The [EFBIG] error is added as part of the large file support extensions.

29049 The [ENXIO] optional error condition is added.

29050 The RETURN VALUE section is updated to note that the error indicator shall be set for the
29051 stream. This is for alignment with the ISO/IEC 9899:1999 standard.

29052 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/31 is applied, updating the [EAGAIN]
29053 error in the ERRORS section from “the process would be delayed” to “the thread would be
29054 delayed”.

29055 Issue 7

29056 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file
29057 descriptors and streams.

29058 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open
29059 Group Technical Standard, 2006, Extended API Set Part 1.

29060 The EXAMPLES section is revised.

29061 Changes are made related to support for finegrained timestamps.

29062 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0126 [87], XSH/TC1-2008/0127 [79],
29063 and XSH/TC1-2008/0128 [14] are applied.

29064 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0112 [816] and XSH/TC2-2008/0113
29065 [626] are applied.

29066 **NAME**

29067 ffs find first set bit

29068 **SYNOPSIS**

```
29069 XSI       #include <strings.h>
29070       int ffs(int i);
```

29071 **DESCRIPTION**

29072 The `ffs()` function shall find the first bit set (beginning with the least significant bit) in *i*, and
29073 return the index of that bit. Bits are numbered starting at one (the least significant bit).

29074 **RETURN VALUE**

29075 The `ffs()` function shall return the index of the first bit set. If *i* is 0, then `ffs()` shall return 0.

29076 **ERRORS**

29077 No errors are defined.

29078 **EXAMPLES**

29079 None.

29080 **APPLICATION USAGE**

29081 None.

29082 **RATIONALE**

29083 None.

29084 **FUTURE DIRECTIONS**

29085 None.

29086 **SEE ALSO**

29087 XBD [<strings.h>](#)

29088 **CHANGE HISTORY**

29089 First released in Issue 4, Version 2.

29090 **Issue 5**

29091 Moved from X/OPEN UNIX extension to BASE.

29092 **NAME**

29093 fgetc — get a byte from a stream

29094 **SYNOPSIS**

29095 #include <stdio.h>

29096 int fgetc(FILE *stream);

29097 **DESCRIPTION**

29098 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29099 conflict between the requirements described here and the ISO C standard is unintentional. This
 29100 volume of POSIX.1-2017 defers to the ISO C standard.

29101 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is
 29102 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,
 29103 from the input stream pointed to by *stream*, and advance the associated file position indicator for
 29104 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple
 29105 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

29106 CX The *fgetc()* function may mark the last data access timestamp of the file associated with *stream*
 29107 for update. The last data access timestamp shall be marked for update by the first successful
 29108 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or
 29109 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

29110 **RETURN VALUE**

29111 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to
 29112 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the
 29113 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,
 29114 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to
 29115 indicate the error.

29116 **ERRORS**29117 The *fgetc()* function shall fail if data needs to be read and:

29118 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
 29119 the thread would be delayed in the *fgetc()* operation.

29120 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
 29121 reading.

29122 CX [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 29123 was transferred.

29124 CX [EIO] A physical I/O error has occurred, or the process is in a background process
 29125 group attempting to read from its controlling terminal, and either the calling
 29126 thread is blocking SIGTIN or the process is ignoring SIGTIN or the process
 29127 group of the process is orphaned. This error may also be generated for
 29128 implementation-defined reasons.

29129 CX [EOVERFLOW] The file is a regular file and an attempt was made to read at or beyond the
 29130 offset maximum associated with the corresponding stream.

29131 The *fgetc()* function may fail if:

29132 CX [ENOMEM] Insufficient storage space is available.

29133 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
 29134 capabilities of the device.

29135 **EXAMPLES**

29136 None.

29137 **APPLICATION USAGE**

29138 If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared
 29139 against the integer constant EOF, the comparison may never succeed, because sign-extension of
 29140 a variable of type **char** on widening to integer is implementation-defined.

29141 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
 29142 end-of-file condition.

29143 **RATIONALE**

29144 None.

29145 **FUTURE DIRECTIONS**

29146 None.

29147 **SEE ALSO**29148 [Section 2.5](#) (on page 495), *feof()*, *ferror()*, *fgets()*, *fread()*, *fscanf()*, *getchar()*, *getc()*, *gets()*, *ungetc()*29149 XBD [<stdio.h>](#)29150 **CHANGE HISTORY**

29151 First released in Issue 1. Derived from Issue 1 of the SVID.

29152 **Issue 5**

29153 Large File Summit extensions are added.

29154 **Issue 6**

29155 Extensions beyond the ISO C standard are marked.

29156 The following new requirements on POSIX implementations derive from alignment with the
 29157 Single UNIX Specification:

29158 The [EIO] and [EOVERFLOW] mandatory error conditions are added.

29159 The [ENOMEM] and [ENXIO] optional error conditions are added.

29160 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

29161 The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the
 29162 input stream is not set.

29163 The RETURN VALUE section is updated to note that the error indicator shall be set for the
 29164 stream.

29165 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/32 is applied, updating the [EAGAIN]
 29166 error in the ERRORS section from “the process would be delayed” to “the thread would be
 29167 delayed”.

29168 **Issue 7**

29169 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark
 29170 the last data access timestamp for update.

29171 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0129 [79] and XSH/TC1-2008/0130
 29172 [14] are applied.

29173 **NAME**29174 `fgetpos` — get current file position information29175 **SYNOPSIS**29176 `#include <stdio.h>`29177 `int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`29178 **DESCRIPTION**

29179 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29180 conflict between the requirements described here and the ISO C standard is unintentional. This
 29181 volume of POSIX.1-2017 defers to the ISO C standard.

29182 The `fgetpos()` function shall store the current values of the parse state (if any) and file position
 29183 indicator for the stream pointed to by `stream` in the object pointed to by `pos`. The value stored
 29184 contains unspecified information usable by `fsetpos()` for repositioning the stream to its position
 29185 at the time of the call to `fgetpos()`.

29186 The `fgetpos()` function shall not change the setting of `errno` if successful.

29187 **RETURN VALUE**

29188 Upon successful completion, `fgetpos()` shall return 0; otherwise, it shall return a non-zero value
 29189 and set `errno` to indicate the error.

29190 **ERRORS**

29191 The `fgetpos()` function shall fail if:

29192 CX [EBADF] The file descriptor underlying `stream` is not valid.

29193 CX [EOVERFLOW] The current value of the file position cannot be represented correctly in an
 29194 object of type `fpos_t`.

29195 CX [ESPIPE] The file descriptor underlying `stream` is associated with a pipe, FIFO, or socket.

29196 **EXAMPLES**

29197 None.

29198 **APPLICATION USAGE**

29199 None.

29200 **RATIONALE**

29201 None.

29202 **FUTURE DIRECTIONS**

29203 None.

29204 **SEE ALSO**

29205 [Section 2.5](#) (on page 495), `fopen()`, `ftell()`, `rewind()`, `ungetc()`

29206 XBD `<stdio.h>`

29207 **CHANGE HISTORY**

29208 First released in Issue 4. Derived from the ISO C standard.

29209 **Issue 5**

29210 Large File Summit extensions are added.

29211 **Issue 6**

29212 Extensions beyond the ISO C standard are marked.

29213 The following new requirements on POSIX implementations derive from alignment with the
29214 Single UNIX Specification:

29215 The [EBADF] and [ESPIPE] optional error conditions are added.

29216 An additional [ESPIPE] error condition is added for sockets.

29217 The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.

29218 **Issue 7**

29219 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0131 [105], XSH/TC1-2008/0132 [122],
29220 and XSH/TC1-2008/0133 [14] are applied.

29221 **NAME**

29222 fgets — get a string from a stream

29223 **SYNOPSIS**

29224 #include <stdio.h>

29225 char *fgets(char *restrict s, int n, FILE *restrict stream);

29226 **DESCRIPTION**

29227 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29228 conflict between the requirements described here and the ISO C standard is unintentional. This
 29229 volume of POSIX.1-2017 defers to the ISO C standard.

29230 The *fgets()* function shall read bytes from *stream* into the array pointed to by *s* until *n*−1 bytes are
 29231 read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered. A
 29232 null byte shall be written immediately after the last byte read into the array. If the end-of-file
 29233 condition is encountered before any bytes are read, the contents of the array pointed to by *s* shall
 29234 not be changed.

29235 CX The *fgets()* function may mark the last data access timestamp of the file associated with *stream*
 29236 for update. The last data access timestamp shall be marked for update by the first successful
 29237 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or
 29238 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

29239 **RETURN VALUE**

29240 Upon successful completion, *fgets()* shall return *s*. If the stream is at end-of-file, the end-of-file
 29241 indicator for the stream shall be set and *fgets()* shall return a null pointer. If a read error occurs,
 29242 CX the error indicator for the stream shall be set, *fgets()* shall return a null pointer, and shall set
 29243 *errno* to indicate the error.

29244 **ERRORS**29245 Refer to *fgetc()*.29246 **EXAMPLES**29247 **Reading Input**

29248 The following example uses *fgets()* to read lines of input. It assumes that the file it is reading is a
 29249 text file and that lines in this text file are no longer than 16384 (or {LINE_MAX} if it is less than
 29250 16384 on the implementation where it is running) bytes long. (Note that the standard utilities
 29251 have no line length limit if *sysconf(_SC_LINE_MAX)* returns −1 without setting *errno*. This
 29252 example assumes that *sysconf(_SC_LINE_MAX)* will not fail.)

```

29253         #include <limits.h>
29254         #include <stdio.h>
29255         #include <unistd.h>
29256         #define MYLIMIT 16384

29257         char *line;
29258         int line_max;
29259         if (LINE_MAX >= MYLIMIT) {
29260                 // Use maximum line size of MYLIMIT. If LINE_MAX is
29261                 // bigger than our limit, sysconf() cannot report a
29262                 // smaller limit.
29263                 line_max = MYLIMIT;
29264         } else {
29265                 long limit = sysconf(_SC_LINE_MAX);
29266                 line_max = (limit < 0 || limit > MYLIMIT) ? MYLIMIT : (int)limit;

```

```

29267     }
29268     // line_max + 1 leaves room for the null byte added by fgets().
29269     line = malloc(line_max + 1);
29270     if (line == NULL) {
29271         // out of space
29272         ...
29273         return error;
29274     }
29275     while (fgets(line, line_max + 1, fp) != NULL) {
29276         // Verify that a full line has been read ...
29277         // If not, report an error or prepare to treat the
29278         // next time through the loop as a read of a
29279         // continuation of the current line.
29280         ...
29281         // Process line ...
29282         ...
29283     }
29284     free(line);
29285     ...

```

29286 **APPLICATION USAGE**
29287 None.

29288 **RATIONALE**
29289 None.

29290 **FUTURE DIRECTIONS**
29291 None.

29292 **SEE ALSO**
29293 [Section 2.5](#) (on page 495), [fgets\(\)](#), [fgetc\(\)](#), [fopen\(\)](#), [fread\(\)](#), [fscanf\(\)](#), [getc\(\)](#), [getchar\(\)](#), [getdelim\(\)](#), [gets\(\)](#),
29294 [ungetc\(\)](#)
29295 XBD [<stdio.h>](#)

29296 **CHANGE HISTORY**
29297 First released in Issue 1. Derived from Issue 1 of the SVID.

29298 **Issue 6**
29299 Extensions beyond the ISO C standard are marked.
29300 The prototype for `fgets()` is changed for alignment with the ISO/IEC 9899:1999 standard.

29301 **Issue 7**
29302 Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark
29303 the last data access timestamp for update.
29304 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0134 [182] and XSH/TC1-2008/0135
29305 [14] are applied.
29306 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0114 [468] is applied.

29307 **NAME**29308 `fgetwc` — get a wide-character code from a stream29309 **SYNOPSIS**29310 `#include <stdio.h>`29311 `#include <wchar.h>`29312 `wint_t fgetwc(FILE *stream);`29313 **DESCRIPTION**

29314 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29315 conflict between the requirements described here and the ISO C standard is unintentional. This
 29316 volume of POSIX.1-2017 defers to the ISO C standard.

29317 The `fgetwc()` function shall obtain the next character (if present) from the input stream pointed to
 29318 by `stream`, convert that to the corresponding wide-character code, and advance the associated file
 29319 position indicator for the stream (if defined).

29320 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

29321 CX The `fgetwc()` function may mark the last data access timestamp of the file associated with `stream`
 29322 for update. The last data access timestamp shall be marked for update by the first successful
 29323 execution of `fgetwc()`, `fgetws()`, `fwscanf()`, `getwc()`, `getwchar()`, `vfwscanf()`, `vfwscanf()`, or `wscanf()`
 29324 using `stream` that returns data not supplied by a prior call to `ungetwc()`.

29325 The `fgetwc()` function shall not change the setting of `errno` if successful.

29326 **RETURN VALUE**

29327 Upon successful completion, the `fgetwc()` function shall return the wide-character code of the
 29328 character read from the input stream pointed to by `stream` converted to a type `wint_t`. If the end-
 29329 of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for
 29330 the stream shall be set and `fgetwc()` shall return WEOF. If a read error occurs, the error indicator
 29331 CX for the stream shall be set, `fgetwc()` shall return WEOF, and shall set `errno` to indicate the error. If
 29332 an encoding error occurs, the error indicator for the stream shall be set, `fgetwc()` shall return
 29333 WEOF, and shall set `errno` to indicate the error.

29334 **ERRORS**

29335 The `fgetwc()` function shall fail if data needs to be read and:

29336 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and
 29337 the thread would be delayed in the `fgetwc()` operation.

29338 CX **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for
 29339 reading.

29340 **[EILSEQ]** The data obtained from the input stream does not form a valid character.

29341 CX **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data
 29342 was transferred.

29343 CX **[EIO]** A physical I/O error has occurred, or the process is in a background process
 29344 group attempting to read from its controlling terminal, and either the calling
 29345 thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process
 29346 group of the process is orphaned. This error may also be generated for
 29347 implementation-defined reasons.

29348 CX **[EOVERFLOW]** The file is a regular file and an attempt was made to read at or beyond the
 29349 offset maximum associated with the corresponding stream.

29350 The *fgetwc()* function may fail if:

29351 CX [ENOMEM] Insufficient storage space is available.

29352 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
29353 capabilities of the device.

29354 EXAMPLES

29355 None.

29356 APPLICATION USAGE

29357 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
29358 end-of-file condition.

29359 RATIONALE

29360 None.

29361 FUTURE DIRECTIONS

29362 None.

29363 SEE ALSO

29364 [Section 2.5](#) (on page 495), *feof()*, *ferror()*, *fopen()*

29365 XBD [<stdio.h>](#), [<wchar.h>](#)

29366 CHANGE HISTORY

29367 First released in Issue 4. Derived from the MSE working draft.

29368 Issue 5

29369 The Optional Header (OH) marking is removed from [<stdio.h>](#).

29370 Large File Summit extensions are added.

29371 Issue 6

29372 Extensions beyond the ISO C standard are marked.

29373 The following new requirements on POSIX implementations derive from alignment with the
29374 Single UNIX Specification:

29375 The [EIO] and [Eoverflow] mandatory error conditions are added.

29376 The [ENOMEM] and [ENXIO] optional error conditions are added.

29377 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/33 is applied, updating the [EAGAIN]
29378 error in the ERRORS section from “the process would be delayed” to “the thread would be
29379 delayed”.

29380 Issue 7

29381 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

29382 Changes are made related to support for finegrained timestamps.

29383 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0136 [105], XSH/TC1-2008/0137 [79],
29384 and XSH/TC1-2008/0138 [14] are applied.

29385 **NAME**

29386 fgetws — get a wide-character string from a stream

29387 **SYNOPSIS**

29388 #include <stdio.h>

29389 #include <wchar.h>

29390 wchar_t *fgetws(wchar_t *restrict ws, int n,

29391 FILE *restrict stream);

29392 **DESCRIPTION**

29393 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29394 conflict between the requirements described here and the ISO C standard is unintentional. This
 29395 volume of POSIX.1-2017 defers to the ISO C standard.

29396 The *fgetws()* function shall read characters from the *stream*, convert these to the corresponding
 29397 wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*−1 characters are
 29398 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is
 29399 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character
 29400 code.

29401 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

29402 CX The *fgetws()* function may mark the last data access timestamp of the file associated with *stream*
 29403 for update. The last data access timestamp shall be marked for update by the first successful
 29404 execution of *fgetwc()*, *fgetws()*, *fwscanf()*, *getwc()*, *getwchar()*, *vwscanf()*, *wscanf()*
 29405 using *stream* that returns data not supplied by a prior call to *ungetwc()*.

29406 **RETURN VALUE**

29407 Upon successful completion, *fgetws()* shall return *ws*. If the end-of-file indicator for the stream is
 29408 set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and
 29409 *fgetws()* shall return a null pointer. If a read error occurs, the error indicator for the stream shall
 29410 CX be set, *fgetws()* shall return a null pointer, and shall set *errno* to indicate the error.

29411 **ERRORS**29412 Refer to *fgetwc()*.29413 **EXAMPLES**

29414 None.

29415 **APPLICATION USAGE**

29416 None.

29417 **RATIONALE**

29418 None.

29419 **FUTURE DIRECTIONS**

29420 None.

29421 **SEE ALSO**29422 Section 2.5 (on page 495), *fopen()*, *fread()*

29423 XBD <stdio.h>, <wchar.h>

29424 **CHANGE HISTORY**

29425 First released in Issue 4. Derived from the MSE working draft.

29426 **Issue 5**

29427 The Optional Header (OH) marking is removed from `<stdio.h>`.

29428 **Issue 6**

29429 Extensions beyond the ISO C standard are marked.

29430 The prototype for `fgetws()` is changed for alignment with the ISO/IEC 9899:1999 standard.

29431 **Issue 7**

29432 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

29433 Changes are made related to support for finegrained timestamps.

29434 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0139 [14] is applied.

29435 **NAME**

29436 fileno — map a stream pointer to a file descriptor

29437 **SYNOPSIS**

```
29438 CX #include <stdio.h>
29439 int fileno(FILE *stream);
```

29440 **DESCRIPTION**

29441 The *fileno()* function shall return the integer file descriptor associated with the stream pointed to
 29442 by *stream*.

29443 **RETURN VALUE**

29444 Upon successful completion, *fileno()* shall return the integer value of the file descriptor
 29445 associated with *stream*. Otherwise, the value -1 shall be returned and *errno* set to indicate the
 29446 error.

29447 **ERRORS**29448 The *fileno()* function shall fail if:

29449 [EBADF] The stream is not associated with a file.

29450 The *fileno()* function may fail if:29451 [EBADF] The file descriptor underlying *stream* is not a valid file descriptor.29452 **EXAMPLES**

29453 None.

29454 **APPLICATION USAGE**

29455 None.

29456 **RATIONALE**

29457 Without some specification of which file descriptors are associated with these streams, it is
 29458 impossible for an application to set up the streams for another application it starts with *fork()*
 29459 and *exec*. In particular, it would not be possible to write a portable version of the *sh* command
 29460 interpreter (although there may be other constraints that would prevent that portability).

29461 **FUTURE DIRECTIONS**

29462 None.

29463 **SEE ALSO**29464 [Section 2.5.1](#) (on page 497), *dirfd()*, *fdopen()*, *fopen()*, *stdin*29465 XBD [<stdio.h>](#)29466 **CHANGE HISTORY**

29467 First released in Issue 1. Derived from Issue 1 of the SVID.

29468 **Issue 6**

29469 The following new requirements on POSIX implementations derive from alignment with the
 29470 Single UNIX Specification:

29471 The [EBADF] optional error condition is added.

29472 **Issue 7**

29473 SD5-XBD-ERN-99 is applied, changing the definition of the [EBADF] error.

29474 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0115 [589] is applied.

29475 **NAME**

29476 flockfile, ftrylockfile, funlockfile ‡'stdio locking functions

29477 **SYNOPSIS**

```
29478 CX #include <stdio.h>
29479 void flockfile(FILE *file);
29480 int ftrylockfile(FILE *file);
29481 void funlockfile(FILE *file);
```

29482 **DESCRIPTION**

29483 These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects.
 29484 These functions can be used by a thread to delineate a sequence of I/O statements that are
 29485 executed as a unit.

29486 The *flockfile()* function shall acquire for a thread ownership of a (**FILE ***) object.

29487 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE ***) object if the object is
 29488 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

29489 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is
 29490 undefined if a thread other than the current owner calls the *funlockfile()* function.

29491 The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This
 29492 count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is
 29493 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***)
 29494 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and
 29495 the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread
 29496 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall
 29497 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*)
 29498 and *funlockfile()* to be nested.

29499 All functions that reference (**FILE ***) objects, except those with names ending in *_unlocked*, shall
 29500 behave as if they use *flockfile()* and *funlockfile()* internally to obtain ownership of these (**FILE ***)
 29501 objects.

29502 **RETURN VALUE**

29503 None for *flockfile()* and *funlockfile()*.

29504 The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock
 29505 cannot be acquired.

29506 **ERRORS**

29507 No errors are defined.

29508 **EXAMPLES**

29509 None.

29510 **APPLICATION USAGE**

29511 Applications using these functions may be subject to priority inversion, as discussed in XBD
 29512 [Section 3.291](#) (on page 80).

29513 A call to *exit()* can block until locked streams are unlocked because a thread having ownership
 29514 of a (**FILE***) object blocks all function calls that reference that (**FILE***) object (except those with
 29515 names ending in *_unlocked*) from other threads, including calls to *exit()*.

29516 **RATIONALE**

29517 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each
29518 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock,
29519 analogous to *pthread_mutex_trylock()*.

29520 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.
29521 This both provides thread-safety of these functions without requiring a second level of internal
29522 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

29523 Application developers and implementors should be aware that there are potential deadlock
29524 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested
29525 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of
29526 implementation-defined line-buffered output streams to be flushed. If two threads each hold the
29527 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring
29528 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can
29529 typically be avoided by acquiring locks on input streams before locks on output streams if a
29530 thread would be acquiring both.

29531 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*
29532 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where
29533 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the
29534 `*_unlocked` functions/macros. This moves the cost/performance tradeoff to the optimal point.

29535 **FUTURE DIRECTIONS**

29536 None.

29537 **SEE ALSO**

29538 [*exit\(\)*](#), [*getc_unlocked\(\)*](#)

29539 XBD [Section 3.291](#) (on page 80), [`<stdio.h>`](#)

29540 **CHANGE HISTORY**

29541 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

29542 **Issue 6**

29543 These functions are marked as part of the Thread-Safe Functions option.

29544 **Issue 7**

29545 The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions are moved from the Thread-Safe Functions
29546 option to the Base.

29547 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0140 [118] is applied.

29548 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0116 [611] is applied.

29549 **NAME**

29550 floor, floorf, floorl ‡'floor function

29551 **SYNOPSIS**

```
29552 #include <math.h>
29553 double floor(double x);
29554 float floorf(float x);
29555 long double floorl(long double x);
```

29556 **DESCRIPTION**

29557 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29558 conflict between the requirements described here and the ISO C standard is unintentional. This
 29559 volume of POSIX.1-2017 defers to the ISO C standard.

29560 These functions shall compute the largest integral value not greater than x .

29561 **RETURN VALUE**

29562 MX The result shall have the same sign as x .

29563 Upon successful completion, these functions shall return the largest integral value not greater
 29564 than x , expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the
 29565 function.

29566 MX If x is NaN, a NaN shall be returned.

29567 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

29568 **ERRORS**

29569 No errors are defined.

29570 **EXAMPLES**

29571 None.

29572 **APPLICATION USAGE**

29573 The integral value returned by these functions might not be expressible as an **intmax_t**. The
 29574 return value should be tested before assigning it to an integer type to avoid the undefined
 29575 results of an integer overflow.

29576 These functions may raise the inexact floating-point exception if the result differs in value from
 29577 the argument.

29578 **RATIONALE**

29579 None.

29580 **FUTURE DIRECTIONS**

29581 None.

29582 **SEE ALSO**

29583 [ceil\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

29584 [Section 4.20](#) (on page 117), [<math.h>](#)

29585 **CHANGE HISTORY**

29586 First released in Issue 1. Derived from Issue 1 of the SVID.

29587 **Issue 5**

29588 The DESCRIPTION is updated to indicate how an application should check for an error. This
 29589 text was previously published in the APPLICATION USAGE section.

29590 **Issue 6**

29591 The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

29592 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
29593 revised to align with the ISO/IEC 9899:1999 standard.

29594 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
29595 marked.

29596 **Issue 7**

29597 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0141 [346] is applied.

29598 **NAME**29599 `fma, fmaf, fmal` ‡floating-point multiply-add29600 **SYNOPSIS**

```
29601 #include <math.h>
29602 double fma(double x, double y, double z);
29603 float fmaf(float x, float y, float z);
29604 long double fmal(long double x, long double y, long double z);
```

29605 **DESCRIPTION**

29606 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29607 conflict between the requirements described here and the ISO C standard is unintentional. This
 29608 volume of POSIX.1-2017 defers to the ISO C standard.

29609 These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute
 29610 the value (as if) to infinite precision and round once to the result format, according to the
 29611 rounding mode characterized by the value of `FLT_ROUNDS`.

29612 An application wishing to check for error situations should set `errno` to zero and call
 29613 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 29614 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 29615 zero, an error has occurred.

29616 **RETURN VALUE**

29617 Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary
 29618 operation.

29619 MX If the result overflows or underflows, a range error may occur. On systems that support the IEC
 29620 60559 Floating-Point option, if the result overflows a range error shall occur.

29621 If x or y are NaN, a NaN shall be returned.

29622 If x multiplied by y is an exact infinity and z is also an infinity but with the opposite sign, a
 29623 domain error shall occur, and either a NaN (if supported), or an implementation-defined value
 29624 shall be returned.

29625 If one of x and y is infinite, the other is zero, and z is not a NaN, a domain error shall occur, and
 29626 either a NaN (if supported), or an implementation-defined value shall be returned.

29627 If one of x and y is infinite, the other is zero, and z is a NaN, a NaN shall be returned and a
 29628 domain error may occur.

29629 If $x*y$ is not $0*\text{Inf}$ nor $\text{Inf}*0$ and z is a NaN, a NaN shall be returned.

29630 **ERRORS**

29631 These functions shall fail if:

29632 MX **Domain Error** The value of $x*y+z$ is invalid, or the value $x*y$ is invalid and z is not a NaN.
 29633 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 29634 then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling`
 29635 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception
 29636 shall be raised.

29637 MX **Range Error** The result overflows.
 29638 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 29639 then `errno` shall be set to [ERANGE]. If the integer expression
 29640 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the overflow
 29641 floating-point exception shall be raised.

29642 These functions may fail if:

29643	MX	Domain Error	The value $x*y$ is invalid and z is a NaN.
29644			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
29645			then <i>errno</i> shall be set to [EDOM]. If the integer expression (<i>math_errhandling</i>
29646			& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
29647			shall be raised.
29648		Range Error	The result underflows.
29649			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
29650			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
29651			(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the underflow
29652			floating-point exception shall be raised.
29653		Range Error	The result overflows.
29654			If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
29655			then <i>errno</i> shall be set to [ERANGE]. If the integer expression
29656			(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
29657			floating-point exception shall be raised.

29658 EXAMPLES

29659 None.

29660 APPLICATION USAGE

29661 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
29662 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

29663 RATIONALE

29664 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its
29665 unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT
29666 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees
29667 its use where desired. Many current machines provide hardware floating multiply-add
29668 instructions; software implementation can be used for others.

29669 FUTURE DIRECTIONS

29670 None.

29671 SEE ALSO

29672 [fclearexcept\(\)](#), [fetestexcept\(\)](#)

29673 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

29674 CHANGE HISTORY

29675 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

29676 Issue 7

29677 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied,
29678 adding a “may fail” range error for non-MX systems.

29679 **NAME**

29680 fmax, fmaxf, fmaxl ‡determine maximum numeric value of two floating-point numbers

29681 **SYNOPSIS**

29682 #include <math.h>

29683 double fmax(double x, double y);

29684 float fmaxf(float x, float y);

29685 long double fmaxl(long double x, long double y);

29686 **DESCRIPTION**29687 CX The functionality described on this reference page is aligned with the ISO C standard. Any
29688 conflict between the requirements described here and the ISO C standard is unintentional. This
29689 volume of POSIX.1-2017 defers to the ISO C standard.29690 MX These functions shall determine the maximum numeric value of their arguments. NaN
29691 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
29692 then these functions shall choose the numeric value.29693 **RETURN VALUE**29694 Upon successful completion, these functions shall return the maximum numeric value of their
29695 arguments.

29696 MX If just one argument is a NaN, the other argument shall be returned.

29697 If *x* and *y* are NaN, a NaN shall be returned.29698 **ERRORS**

29699 No errors are defined.

29700 **EXAMPLES**

29701 None.

29702 **APPLICATION USAGE**

29703 None.

29704 **RATIONALE**

29705 None.

29706 **FUTURE DIRECTIONS**

29707 None.

29708 **SEE ALSO**29709 *fdim()*, *fmin()*

29710 XBD <math.h>

29711 **CHANGE HISTORY**

29712 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

29713 **Issue 7**

29714 Austin Group Interpretation 1003.1-2001 #007 is applied.

29715 **NAME**

29716 fmemopen — open a memory buffer stream

29717 **SYNOPSIS**

```
29718 CX #include <stdio.h>
29719 FILE *fmemopen(void *restrict buf, size_t size,
29720 const char *restrict mode);
```

29721 **DESCRIPTION**

29722 The *fmemopen()* function shall associate the buffer given by the *buf* and *size* arguments with a
 29723 stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least *size*
 29724 bytes long.

29725 The *mode* argument points to a string. If the string is one of the following, the stream shall be
 29726 opened in the indicated mode. Otherwise, the behavior is undefined.

29727 *r* Open the stream for reading.

29728 *w* Open the stream for writing.

29729 *a* Append; open the stream for writing at the first null byte.

29730 *r+* Open the stream for update (reading and writing).

29731 *w+* Open the stream for update (reading and writing). Truncate the buffer contents.

29732 *a+* Append; open the stream for update (reading and writing); the initial position is at the
 29733 first null byte.

29734 Implementations shall accept all mode strings allowed by *fopen()*, but the use of the character
 29735 'b' shall produce implementation-defined results, where the resulting **FILE *** need not behave
 29736 the same as if 'b' were omitted.

29737 If a null pointer is specified as the *buf* argument, *fmemopen()* shall allocate *size* bytes of memory
 29738 as if by a call to *malloc()*. This buffer shall be automatically freed when the stream is closed.
 29739 Because this feature is only useful when the stream is opened for updating (because there is no
 29740 way to get a pointer to the buffer) the *fmemopen()* call may fail if the *mode* argument does not
 29741 include a '+ '.

29742 The stream shall maintain a current position in the buffer. This position shall be initially set to
 29743 either the beginning of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a*
 29744 modes). If no null byte is found in append mode, the initial position shall be set to one byte after
 29745 the end of the buffer.

29746 If *buf* is a null pointer, the initial position shall always be set to the beginning of the buffer.

29747 The stream shall also maintain the size of the current buffer contents; use of *fseek()* or *fseeko()* on
 29748 the stream with *SEEK_END* shall seek relative to this size. For modes *r* and *r+* the size shall be
 29749 set to the value given by the *size* argument. For modes *w* and *w+* the initial size shall be zero and
 29750 for modes *a* and *a+* the initial size shall be:

29751 Zero, if *buf* is a null pointer

29752 The position of the first null byte in the buffer, if one is found

29753 The value of the *size* argument, if *buf* is not a null pointer and no null byte is found

29754 A read operation on the stream shall not advance the current buffer position beyond the current
 29755 buffer size. Reaching the buffer size in a read operation shall count as "end-of-file". Null bytes in
 29756 the buffer shall have no special meaning for reads. The read operation shall start at the current

29757 buffer position of the stream.

29758 A write operation shall start either at the current position of the stream (if *mode* has not specified
29759 'a' as the first character) or at the current size of the stream (if *mode* had 'a' as the first
29760 character). If the current position at the end of the write is larger than the current buffer size, the
29761 current buffer size shall be set to the current position. A write operation on the stream shall not
29762 advance the current buffer size beyond the size given in the *size* argument.

29763 When a stream open for writing is flushed or closed, a null byte shall be written at the current
29764 position or at the end of the buffer, depending on the size of the contents. If a stream open for
29765 update is flushed or closed and the last write has advanced the current buffer size, a null byte
29766 shall be written at the end of the buffer if it fits.

29767 An attempt to seek a memory buffer stream to a negative position or to a position larger than the
29768 buffer size given in the *size* argument shall fail.

29769 RETURN VALUE

29770 Upon successful completion, *fmemopen()* shall return a pointer to the object controlling the
29771 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

29772 ERRORS

29773 The *fmemopen()* function shall fail if:

29774 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

29775 The *fmemopen()* function may fail if:

29776 [EINVAL] The value of the *mode* argument is not valid.

29777 [EINVAL] The *buf* argument is a null pointer and the *mode* argument does not include a
29778 '+' character.

29779 [EINVAL] The *size* argument specifies a buffer size of zero and the implementation does
29780 not support this.

29781 [ENOMEM] The *buf* argument is a null pointer and the allocation of a buffer of length *size*
29782 has failed.

29783 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

29784 EXAMPLES

```
29785 #include <stdio.h>
29786 #include <string.h>
29787 static char buffer[] = "foobar";
29788
29789 int
29790 main (void)
29791 {
29792     int ch;
29793     FILE *stream;
29794
29795     stream = fmemopen(buffer, strlen (buffer), "r");
29796     if (stream == NULL)
29797         /* handle error */;
29798
29799     while ((ch = fgetc(stream)) != EOF)
29800         printf("Got %c\n", ch);
29801
29802     fclose(stream);
```

```
29799         return (0);
29800     }
```

29801 This program produces the following output:

```
29802     Got f
29803     Got o
29804     Got o
29805     Got b
29806     Got a
29807     Got r
```

29808 APPLICATION USAGE

29809 None.

29810 RATIONALE

29811 This interface has been introduced to eliminate many of the errors encountered in the
29812 construction of strings, notably overflowing of strings. This interface prevents overflow.

29813 FUTURE DIRECTIONS

29814 A future version of this standard may mandate specific behavior when the *mode* argument
29815 includes 'b'.

29816 A future version of this standard may require support of zero-length buffer streams explicitly.

29817 SEE ALSO

29818 [*fdopen\(\)*](#), [*fopen\(\)*](#), [*freopen\(\)*](#), [*fseek\(\)*](#), [*malloc\(\)*](#), [*open_memstream\(\)*](#)

29819 XBD [**<stdio.h>**](#)

29820 CHANGE HISTORY

29821 First released in Issue 7.

29822 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0142 [461], XSH/TC1-2008/0143 [396],
29823 XSH/TC1-2008/0144 [396], XSH/TC1-2008/0145 [461], XSH/TC1-2008/0146 [461],
29824 XSH/TC1-2008/0147 [461], XSH/TC1-2008/0148 [461], XSH/TC1-2008/0149 [461], and
29825 XSH/TC1-2008/0150 [396] are applied.

29826 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0117 [587], XSH/TC2-2008/0118
29827 [586,818], and XSH/TC2-2008/0119 [818] are applied.

29828 **NAME**

29829 fmin, fminf, fminl ‡determine minimum numeric value of two floating-point numbers

29830 **SYNOPSIS**

29831 #include <math.h>

29832 double fmin(double x, double y);

29833 float fminf(float x, float y);

29834 long double fminl(long double x, long double y);

29835 **DESCRIPTION**29836 CX The functionality described on this reference page is aligned with the ISO C standard. Any
29837 conflict between the requirements described here and the ISO C standard is unintentional. This
29838 volume of POSIX.1-2017 defers to the ISO C standard.29839 MX These functions shall determine the minimum numeric value of their arguments. NaN
29840 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
29841 then these functions shall choose the numeric value.29842 **RETURN VALUE**29843 Upon successful completion, these functions shall return the minimum numeric value of their
29844 arguments.

29845 MX If just one argument is a NaN, the other argument shall be returned.

29846 If *x* and *y* are NaN, a NaN shall be returned.29847 **ERRORS**

29848 No errors are defined.

29849 **EXAMPLES**

29850 None.

29851 **APPLICATION USAGE**

29852 None.

29853 **RATIONALE**

29854 None.

29855 **FUTURE DIRECTIONS**

29856 None.

29857 **SEE ALSO**29858 *fdim()*, *fmax()*

29859 XBD <math.h>

29860 **CHANGE HISTORY**

29861 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

29862 **Issue 7**

29863 Austin Group Interpretation 1003.1-2001 #008 is applied.

29864 **NAME**29865 `fmod, fmodf, fmodl` — floating-point remainder value function29866 **SYNOPSIS**

```
29867 #include <math.h>
29868 double fmod(double x, double y);
29869 float fmodf(float x, float y);
29870 long double fmodl(long double x, long double y);
```

29871 **DESCRIPTION**

29872 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29873 conflict between the requirements described here and the ISO C standard is unintentional. This
 29874 volume of POSIX.1-2017 defers to the ISO C standard.

29875 These functions shall return the floating-point remainder of the division of x by y .

29876 An application wishing to check for error situations should set *errno* to zero and call
 29877 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 29878 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 29879 zero, an error has occurred.

29880 **RETURN VALUE**

29881 These functions shall return the value $x - i * y$, for some integer i such that, if y is non-zero, the
 29882 result has the same sign as x and magnitude less than the magnitude of y .

29883 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 29884 MXX and *fmod()*, *modf()*, and *fmodl()* shall return 0.0, or (if the IEC 60559 Floating-Point option is not
 29885 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 29886 FLT_MIN, and LDBL_MIN, respectively.

29887 MX If x or y is NaN, a NaN shall be returned, and none of the conditions below shall be considered.

29888 If y is zero, a domain error shall occur, and a NaN shall be returned.

29889 If x is infinite, a domain error shall occur, and a NaN shall be returned.

29890 If x is ± 0 and y is not zero, ± 0 shall be returned.

29891 If x is not infinite and y is $\pm \text{Inf}$, x shall be returned.

29892 MXX If the correct value would cause underflow, and is representable, a range error may occur and
 29893 the correct value shall be returned.

29894 **ERRORS**

29895 These functions shall fail if:

29896 MX **Domain Error** The x argument is infinite or y is zero.
 29897 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 29898 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 29899 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 29900 shall be raised.

29901 These functions may fail if:

29902 **Range Error** The result underflows.
 29903 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 29904 then *errno* shall be set to [ERANGE]. If the integer expression
 29905 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 29906 floating-point exception shall be raised.

29907 **EXAMPLES**

29908 None.

29909 **APPLICATION USAGE**

29910 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
29911 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

29912 **RATIONALE**

29913 None.

29914 **FUTURE DIRECTIONS**

29915 None.

29916 **SEE ALSO**29917 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)29918 [Section 4.20](#) (on page 117), [<math.h>](#)29919 **CHANGE HISTORY**

29920 First released in Issue 1. Derived from Issue 1 of the SVID.

29921 **Issue 5**

29922 The DESCRIPTION is updated to indicate how an application should check for an error. This
29923 text was previously published in the APPLICATION USAGE section.

29924 **Issue 6**29925 The behavior for when the *y* argument is zero is now defined.

29926 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999
29927 standard.

29928 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
29929 revised to align with the ISO/IEC 9899:1999 standard.

29930 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
29931 marked.

29932 **Issue 7**

29933 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0151 [68], XSH/TC1-2008/0152 [320],
29934 and XSH/TC1-2008/0153 [68] are applied.

29935 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0120 [605] is applied.

29936 **NAME**29937 `fmtmsg` `†`'display a message in the specified format on standard error and/or a system console29938 **SYNOPSIS**

```
29939 XSI #include <fmtmsg.h>
29940 int fmtmsg(long classification, const char *label, int severity,
29941            const char *text, const char *action, const char *tag);
```

29942 **DESCRIPTION**29943 The `fmtmsg()` function shall display messages in a specified format instead of the traditional
29944 `printf()` function.29945 Based on a message's classification component, `fmtmsg()` shall write a formatted message either
29946 to standard error, to the console, or to both.29947 A formatted message consists of up to five components as defined below. The component
29948 *classification* is not part of a message displayed to the user, but defines the source of the message
29949 and directs the display of the formatted message.

29950 *classification* Contains the sum of identifying values constructed from the constants defined
29951 below. Any one identifier from a subclass may be used in combination with a
29952 single identifier from a different subclass. Two or more identifiers from the
29953 same subclass should not be used together, with the exception of identifiers
29954 from the display subclass. (Both display subclass identifiers may be used so
29955 that messages can be displayed to both standard error and the system
29956 console.)

29957 **Major Classifications**29958 Identifies the source of the condition. Identifiers are: MM_HARD
29959 (hardware), MM_SOFT (software), and MM_FIRM (firmware).29960 **Message Source Subclassifications**29961 Identifies the type of software in which the problem is detected.
29962 Identifiers are: MM_APPL (application), MM_UTIL (utility), and
29963 MM OPSYS (operating system).29964 **Display Subclassifications**29965 Indicates where the message is to be displayed. Identifiers are:
29966 MM_PRINT to display the message on the standard error stream,
29967 MM_CONSOLE to display the message on the system console. One or
29968 both identifiers may be used.29969 **Status Subclassifications**29970 Indicates whether the application can recover from the condition.
29971 Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-
29972 recoverable).29973 An additional identifier, MM_NULLMC, indicates that no classification
29974 component is supplied for the message.29975 *label* Identifies the source of the message. The format is two fields separated by a
29976 <colon>. The first field is up to 10 bytes, the second is up to 14 bytes.29977 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*
29978 are:

29979		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
29980			
29981		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
29982			
29983		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
29984			
29985			
29986		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
29987			
29988		MM_NOSEV	Indicates that no severity level is supplied for the message.
29989	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
29990			
29991			
29992	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
29993			
29994			
29995	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
29996			
29997			

29998 The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg()*
 29999 which message components it is to select when writing messages to standard error. The value of
 30000 *MSGVERB* shall be a <colon>-separated list of optional keywords. Valid keywords are: *label*,
 30001 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the
 30002 component's value is not the component's null value, *fmtmsg()* shall include that component in
 30003 the message when writing the message to standard error. If *MSGVERB* does not include a
 30004 keyword for a message component, that component shall not be included in the display of the
 30005 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the
 30006 null string, if its value is not of the correct format, or if it contains keywords other than the valid
 30007 ones listed above, *fmtmsg()* shall select all components.

30008 *MSGVERB* shall determine which components are selected for display to standard error. All
 30009 message components shall be included in console messages.

30010 RETURN VALUE

30011 The *fmtmsg()* function shall return one of the following values:

30012	MM_OK	The function succeeded.
30013	MM_NOTOK	The function failed completely.
30014	MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.
30015		
30016	MM_NOCON	The function was unable to generate a console message, but otherwise succeeded.
30017		

30018 ERRORS

30019 None.

30020 **EXAMPLES**30021 1. The following example of *fmtmsg()*:30022 `fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",`
30023 `"refer to cat in user's reference manual", "XSI:cat:001")`

30024 produces a complete message in the specified message format:

30025 `XSI:cat: ERROR: illegal option`
30026 `TO FIX: refer to cat in user's reference manual XSI:cat:001`30027 2. When the environment variable *MSGVERB* is set as follows:30028 `MSGVERB=severity:text:action`30029 and Example 1 is used, *fmtmsg()* produces:30030 `ERROR: illegal option`
30031 `TO FIX: refer to cat in user's reference manual`30032 **APPLICATION USAGE**30033 One or more message components may be systematically omitted from messages generated by
30034 an application by using the null value of the argument for that component.30035 **RATIONALE**

30036 None.

30037 **FUTURE DIRECTIONS**

30038 None.

30039 **SEE ALSO**30040 [*fprintf\(\)*](#)30041 XBD [<fmtmsg.h>](#)30042 **CHANGE HISTORY**

30043 First released in Issue 4, Version 2.

30044 **Issue 5**

30045 Moved from X/OPEN UNIX extension to BASE.

30046 **NAME**

30047 fnmatch ‡match a filename string or a pathname

30048 **SYNOPSIS**

30049 #include <fnmatch.h>

30050 int fnmatch(const char *pattern, const char *string, int flags);

30051 **DESCRIPTION**

30052 The *fnmatch()* function shall match patterns as described in XCU [Section 2.13.1](#) (on page 2382)
 30053 and [Section 2.13.2](#) (on page 2383). It checks the string specified by the *string* argument to see if it
 30054 matches the pattern specified by the *pattern* argument.

30055 The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-
 30056 inclusive OR of zero or more of the flags defined in <fnmatch.h>. If the FNM_PATHNAME flag
 30057 is set in *flags*, then a <slash> character ('/') in *string* shall be explicitly matched by a <slash> in
 30058 *pattern*; it shall not be matched by either the <asterisk> or <question-mark> special characters,
 30059 nor by a bracket expression. If the FNM_PATHNAME flag is not set, the <slash> character shall
 30060 be treated as an ordinary character.

30061 If FNM_NOESCAPE is not set in *flags*, a <backslash> character in *pattern* followed by any other
 30062 character shall match that second character in *string*. In particular, "\\\" shall match a
 30063 <backslash> in *string*. If *pattern* ends with an unescaped <backslash>, *fnmatch()* shall return a
 30064 non-zero value (indicating either no match or an error). If FNM_NOESCAPE is set, a
 30065 <backslash> character shall be treated as an ordinary character.

30066 If FNM_PERIOD is set in *flags*, then a leading <period> ('.') in *string* shall match a <period> in
 30067 *pattern*; as described by rule 2 in XCU [Section 2.13.3](#) (on page 2383) where the location of
 30068 “leading” is indicated by the value of FNM_PATHNAME:

30069 If FNM_PATHNAME is set, a <period> is “leading” if it is the first character in *string* or if
 30070 it immediately follows a <slash>.

30071 If FNM_PATHNAME is not set, a <period> is “leading” only if it is the first character of
 30072 *string*.

30073 If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

30074 **RETURN VALUE**

30075 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return 0. If there is no
 30076 match, *fnmatch()* shall return FNM_NOMATCH, which is defined in <fnmatch.h>. If an error
 30077 occurs, *fnmatch()* shall return another non-zero value.

30078 **ERRORS**

30079 No errors are defined.

30080 **EXAMPLES**

30081 None.

30082 **APPLICATION USAGE**

30083 The *fnmatch()* function has two major uses. It could be used by an application or utility that
 30084 needs to read a directory and apply a pattern against each entry. The *find* utility is an example of
 30085 this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that
 30086 need to match strings in a similar manner.

30087 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname* match. The
 30088 default action of this function is to match filename strings, rather than pathnames, since it gives
 30089 no special significance to the <slash> character. With the FNM_PATHNAME flag, *fnmatch()* does
 30090 match pathnames, but without tilde expansion, parameter expansion, or special treatment for a

30091 <period> at the beginning of a filename.

30092 **RATIONALE**

30093 This function replaced the REG_FILENAME flag of *regcomp()* in early proposals of this volume
30094 of POSIX.1-2017. It provides virtually the same functionality as the *regcomp()* and *regexec()*
30095 functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag was
30096 proposed for *regcomp()*, and would have had the opposite effect from FNM_PATHNAME), but
30097 with a simpler function and less system overhead.

30098 **FUTURE DIRECTIONS**

30099 None.

30100 **SEE ALSO**

30101 [glob\(\)](#), [Section 2.6](#)

30102 XBD [<fnmatch.h>](#)

30103 **CHANGE HISTORY**

30104 First released in Issue 4. Derived from the ISO POSIX-2 standard.

30105 **Issue 5**

30106 Moved from POSIX2 C-language Binding to BASE.

30107 **Issue 7**

30108 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0154 [291] and XSH/TC1-2008/0155
30109 [291] are applied.

30110 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0121 [806] is applied.

30111 **NAME**

30112 fopen — open a stream

30113 **SYNOPSIS**

30114 #include <stdio.h>

30115 FILE *fopen(const char *restrict *pathname*, const char *restrict *mode*);30116 **DESCRIPTION**

30117 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 30118 conflict between the requirements described here and the ISO C standard is unintentional. This
 30119 volume of POSIX.1-2017 defers to the ISO C standard.

30120 The *fopen()* function shall open the file whose *pathname* is the string pointed to by *pathname*,
 30121 and associates a stream with it.

30122 The *mode* argument points to a string. If the string is one of the following, the file shall be opened
 30123 in the indicated mode. Otherwise, the behavior is undefined.

30124 *r* or *rb* Open file for reading.

30125 *w* or *wb* Truncate to zero length or create file for writing.

30126 *a* or *ab* Append; open or create file for writing at end-of-file.

30127 *r+* or *rb+* or *r+b* Open file for update (reading and writing).

30128 *w+* or *wb+* or *w+b* Truncate to zero length or create file for update.

30129 *a+* or *ab+* or *a+b* Append; open or create file for update, writing at end-of-file.

30130 CX The character 'b' shall have no effect, but is allowed for ISO C standard conformance. Opening
 30131 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not
 30132 exist or cannot be read.

30133 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all
 30134 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
 30135 calls to *fseek()*.

30136 When a file is opened with update mode ('+' as the second or third character in the *mode*
 30137 argument), both input and output may be performed on the associated stream. However, the
 30138 application shall ensure that output is not directly followed by input without an intervening call
 30139 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly
 30140 followed by output without an intervening call to a file positioning function, unless the input
 30141 operation encounters end-of-file.

30142 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
 30143 interactive device. The error and end-of-file indicators for the stream shall be cleared.

30144 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon
 30145 successful completion, *fopen()* shall mark for update the last data access, last data modification,
 30146 and last file status change timestamps of the file and the last file status change and last data
 30147 modification timestamps of the parent directory.

30148 If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, the
 30149 *fopen()* function shall create a file as if it called the *creat()* function with a value appropriate for
 30150 the *path* argument interpreted from *pathname* and a value of S_IRUSR | S_IWUSR | S_IRGRP |
 30151 S_IWGRP | S_IROTH | S_IWOTH for the *mode* argument.

30152 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,
 30153 *fopen()* shall mark for update the last data modification and last file status change timestamps of

30154 the file.

30155 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the
30156 encoding rule shall be cleared, and the associated *mbstate_t* object shall be set to describe an
30157 initial conversion state.

30158 CX The file descriptor associated with the opened stream shall be allocated and opened as if by a
30159 call to *open()* with the following flags:

<i>fopen()</i> Mode	<i>open()</i> Flags
<i>r</i> or <i>rb</i>	O_RDONLY
<i>w</i> or <i>wb</i>	O_WRONLY O_CREAT O_TRUNC
<i>a</i> or <i>ab</i>	O_WRONLY O_CREAT O_APPEND
<i>r+</i> or <i>rb+</i> or <i>r+b</i>	O_RDWR
<i>w+</i> or <i>wb+</i> or <i>w+b</i>	O_RDWR O_CREAT O_TRUNC
<i>a+</i> or <i>ab+</i> or <i>a+b</i>	O_RDWR O_CREAT O_APPEND

30167 RETURN VALUE

30168 Upon successful completion, *fopen()* shall return a pointer to the object controlling the stream.
30169 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

30170 ERRORS

30171 The *fopen()* function shall fail if:

30172 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
30173 exists and the permissions specified by *mode* are denied, or the file does not
30174 exist and write permission is denied for the parent directory of the file to be
30175 created.

30176 CX [EINTR] A signal was caught during *fopen()*.

30177 CX [EISDIR] The named file is a directory and *mode* requires write access.

30178 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
30179 argument.

30180 CX [EMFILE] All file descriptors available to the process are currently open.

30181 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

30182 CX [ENAMETOOLONG]

30183 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
30184 symbolic link produced an intermediate result with a length that exceeds
30185 {PATH_MAX}.

30186 CX [ENFILE] The maximum allowable number of files is currently open in the system.

30187 CX [ENOENT] The *mode* string begins with 'r' and a component of *pathname* does not name
30188 an existing file, or *mode* begins with 'w' or 'a' and a component of the path
30189 prefix of *pathname* does not name an existing file, or *pathname* is an empty
30190 string.

30191 CX [ENOENT] or [ENOTDIR]

30192 The *pathname* argument contains at least one non-*<slash>* character and ends
30193 with one or more trailing *<slash>* characters. If *pathname* without the trailing
30194 *<slash>* characters would name an existing file, an [ENOENT] error shall not
30195 occur.

30196	CX	[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and the file was to be created.
30197			
30198	CX	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>pathname</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
30199			
30200			
30201			
30202			
30203	CX	[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
30204			
30205	CX	[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type <code>off_t</code> .
30206			
30207	CX	[EROFS]	The named file resides on a read-only file system and <i>mode</i> requires write access.
30208			
30209			The <i>fopen()</i> function may fail if:
30210	CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
30211	CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
30212			
30213	CX	[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.
30214	CX	[ENAMETOOLONG]	
30215			The length of a component of a pathname is longer than {NAME_MAX}.
30216	CX	[ENOMEM]	Insufficient storage space is available.
30217	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.
30218			

30219 EXAMPLES

30220 Opening a File

30221 The following example tries to open the file named **file** for reading. The *fopen()* function returns
 30222 a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the
 30223 file, it just ignores it.

```

30224 #include <stdio.h>
30225 ...
30226 FILE *fp;
30227 ...
30228 void rgrep(const char *file)
30229 {
30230     ...
30231     if ((fp = fopen(file, "r")) == NULL)
30232         return;
30233     ...
30234 }
```

30235 **APPLICATION USAGE**

30236 None.

30237 **RATIONALE**

30238 None.

30239 **FUTURE DIRECTIONS**

30240 None.

30241 **SEE ALSO**30242 [Section 2.5](#) (on page 495), [creat\(\)](#), [fclose\(\)](#), [fdopen\(\)](#), [fmemopen\(\)](#), [freopen\(\)](#), [open_memstream\(\)](#)30243 XBD [<stdio.h>](#)30244 **CHANGE HISTORY**

30245 First released in Issue 1. Derived from Issue 1 of the SVID.

30246 **Issue 5**

30247 Large File Summit extensions are added.

30248 **Issue 6**

30249 Extensions beyond the ISO C standard are marked.

30250 The following new requirements on POSIX implementations derive from alignment with the
30251 Single UNIX Specification:30252 In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open
30253 file description. This change is to support large files.30254 In the ERRORS section, the [Eoverflow] condition is added. This change is to support
30255 large files.

30256 The [ELOOP] mandatory error condition is added.

30257 The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional
30258 error conditions are added.

30259 The normative text is updated to avoid use of the term “must” for application requirements.

30260 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

30261 The prototype for *fopen()* is updated.30262 The DESCRIPTION is updated to note that if the argument *mode* points to a string other
30263 than those listed, then the behavior is undefined.30264 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
30265 [ELOOP] error condition is added.30266 **Issue 7**

30267 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode.

30268 Austin Group Interpretation 1003.1-2001 #143 is applied.

30269 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set
30270 on the open file description.

30271 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

30272 SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a
30273 “may fail” to a “shall fail”.

30274 Changes are made related to support for finegrained timestamps.

30275
30276
30277

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0156 [291,433], XSH/TC1-2008/0157 [146,433], XSH/TC1-2008/0158 [324], and XSH/TC1-2008/0159 [14] are applied.
POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0122 [822] is applied.

30278 **NAME**

30279 fork — create a new process

30280 **SYNOPSIS**

30281 #include <unistd.h>

30282 pid_t fork(void);

30283 **DESCRIPTION**30284 The *fork()* function shall create a new process. The new process (child process) shall be an exact
30285 copy of the calling process (parent process) except as detailed below:

30286 The child process shall have a unique process ID.

30287 The child process ID also shall not match any active process group ID.

30288 The child process shall have a different parent process ID, which shall be the process ID of
30289 the calling process.30290 The child process shall have its own copy of the parent's file descriptors. Each of the
30291 child's file descriptors shall refer to the same open file description with the corresponding
30292 file descriptor of the parent.30293 The child process shall have its own copy of the parent's open directory streams. Each
30294 open directory stream in the child process may share directory stream positioning with the
30295 corresponding directory stream of the parent.

30296 The child process shall have its own copy of the parent's message catalog descriptors.

30297 The child process values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to
30298 0.30299 The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
30300 canceled; see *alarm()*.30301 XSI All *semadj* values shall be cleared.

30302 File locks set by the parent process shall not be inherited by the child process.

30303 The set of signals pending for the child process shall be initialized to the empty set.

30304 XSI Interval timers shall be reset in the child process.

30305 Any semaphores that are open in the parent process shall also be open in the child process.

30306 ML The child process shall not inherit any address space memory locks established by the
30307 parent process via calls to *mlockall()* or *mlock()*.30308 Memory mappings created in the parent shall be retained in the child process.
30309 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE
30310 mappings in the child, and any modifications to the data in these mappings made by the
30311 parent prior to calling *fork()* shall be visible to the child. Any modifications to the data in
30312 MAP_PRIVATE mappings made by the parent after *fork()* returns shall be visible only to
30313 the parent. Modifications to the data in MAP_PRIVATE mappings made by the child shall
30314 be visible only to the child.30315 PS For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit
30316 the policy and priority settings of the parent process during a *fork()* function. For other
30317 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.

- 30318 Per-process timers created by the parent shall not be inherited by the child process.
- 30319 MSG The child process shall have its own copy of the message queue descriptors of the parent.
30320 Each of the message descriptors of the child shall refer to the same open message queue
30321 description as the corresponding message descriptor of the parent.
- 30322 No asynchronous input or asynchronous output operations shall be inherited by the child
30323 process. Any use of asynchronous control blocks created by the parent produces undefined
30324 behavior.
- 30325 A process shall be created with a single thread. If a multi-threaded process calls *fork()*, the
30326 new process shall contain a replica of the calling thread and its entire address space,
30327 possibly including the states of mutexes and other resources. Consequently, to avoid
30328 errors, the child process may only execute async-signal-safe operations until such time as
30329 one of the *exec* functions is called.
- 30330 When the application calls *fork()* from a signal handler and any of the fork handlers
30331 registered by *pthread_atfork()* calls a function that is not async-signal-safe, the behavior is
30332 undefined.
- 30333 OB TRC TRI If the Trace option and the Trace Inherit option are both supported:
30334 If the calling process was being traced in a trace stream that had its inheritance policy set
30335 to *POSIX_TRACE_INHERITED*, the child process shall be traced into that trace stream,
30336 and the child process shall inherit the parent's mapping of trace event names to trace event
30337 type identifiers. If the trace stream in which the calling process was being traced had its
30338 inheritance policy set to *POSIX_TRACE_CLOSE_FOR_CHILD*, the child process shall not
30339 be traced into that trace stream. The inheritance policy is set by a call to the
30340 *posix_trace_attr_setinherited()* function.
- 30341 OB TRC If the Trace option is supported, but the Trace Inherit option is not supported:
30342 The child process shall not be traced into any of the trace streams of its parent process.
- 30343 OB TRC If the Trace option is supported, the child process of a trace controller process shall not
30344 control the trace streams controlled by its parent process.
- 30345 CPT The initial value of the CPU-time clock of the child process shall be set to zero.
- 30346 TCT The initial value of the CPU-time clock of the single thread of the child process shall be set
30347 to zero.
- 30348 All other process characteristics defined by POSIX.1-2017 shall be the same in the parent and
30349 child processes. The inheritance of process characteristics not defined by POSIX.1-2017 is
30350 unspecified by POSIX.1-2017.
- 30351 After *fork()*, both the parent and the child processes shall be capable of executing independently
30352 before either one terminates.
- 30353 **RETURN VALUE**
30354 Upon successful completion, *fork()* shall return 0 to the child process and shall return the
30355 process ID of the child process to the parent process. Both processes shall continue to execute
30356 from the *fork()* function. Otherwise, -1 shall be returned to the parent process, no child process
30357 shall be created, and *errno* shall be set to indicate the error.
- 30358 **ERRORS**
30359 The *fork()* function shall fail if:

30360 [EAGAIN] The system lacked the necessary resources to create another process, or the
 30361 system-imposed limit on the total number of processes under execution
 30362 system-wide or by a single user {CHILD_MAX} would be exceeded.

30363 The *fork()* function may fail if:

30364 [ENOMEM] Insufficient storage space is available.

30365 EXAMPLES

30366 None.

30367 APPLICATION USAGE

30368 None.

30369 RATIONALE

30370 Many historical implementations have timing windows where a signal sent to a process group
 30371 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the
 30372 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of
 30373 pending signals. This volume of POSIX.1-2017 does not require, or even permit, this behavior.
 30374 However, it is pragmatic to expect that problems of this nature may continue to exist in
 30375 implementations that appear to conform to this volume of POSIX.1-2017 and pass available
 30376 verification suites. This behavior is only a consequence of the implementation failing to make
 30377 the interval between signal generation and delivery totally invisible. From the application's
 30378 perspective, a *fork()* call should appear atomic. A signal that is generated prior to the *fork()*
 30379 should be delivered prior to the *fork()*. A signal sent to the process group after the *fork()* should
 30380 be delivered to both parent and child. The implementation may actually initialize internal data
 30381 structures corresponding to the child's set of pending signals to include signals sent to the
 30382 process group during the *fork()*. Since the *fork()* call can be considered as atomic from the
 30383 application's perspective, the set would be initialized as empty and such signals would have
 30384 arrived after the *fork()*; see also <signal.h>.

30385 One approach that has been suggested to address the problem of signal inheritance across *fork()*
 30386 is to add an [EINTR] error, which would be returned when a signal is detected during the call.
 30387 While this is preferable to losing signals, it was not considered an optimal solution. Although it
 30388 is not recommended for this purpose, such an error would be an allowable extension for an
 30389 implementation.

30390 The [ENOMEM] error value is reserved for those implementations that detect and distinguish
 30391 such a condition. This condition occurs when an implementation detects that there is not enough
 30392 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate
 30393 because there can never be enough memory (either primary or secondary storage) to perform
 30394 the operation. Since *fork()* duplicates an existing process, this must be a condition where there is
 30395 sufficient memory for one such process, but not for two. Many historical implementations
 30396 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally
 30397 distinct from [EAGAIN] from the perspective of a conforming application.

30398 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it
 30399 and it should be reserved for the error condition specified there. The condition is not applicable
 30400 on many implementations.

30401 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.
 30402 A system that single-threads processes was clearly not intended and is considered an
 30403 unacceptable "toy implementation" of this volume of POSIX.1-2017. The only objection
 30404 anticipated to the phrase "executing independently" is testability, but this assertion should be
 30405 testable. Such tests require that both the parent and child can block on a detectable action of the
 30406 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be

30407 possible for the system to conform to the intent of this volume of POSIX.1-2017.

30408 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it
30409 occurs or not is not in any practical sense under the control of the application because the
30410 condition is usually a consequence of the user's use of the system, not of the application's code.
30411 Thus, no application can or should rely upon its occurrence under any circumstances, nor
30412 should the exact semantics of what concept of "user" is used be of concern to the application
30413 developer. Validation writers should be cognizant of this limitation.

30414 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread
30415 of control within the same program (which was originally only possible in POSIX by creating a
30416 new process); the other is to create a new process running a different program. In the latter case,
30417 the call to *fork()* is soon followed by a call to one of the *exec* functions.

30418 The general problem with making *fork()* work in a multi-threaded world is what to do with all
30419 of the threads. There are two alternatives. One is to copy all of the threads into the new process.
30420 This causes the programmer or implementation to deal with threads that are suspended on
30421 system calls or that might be about to execute system calls that should not be executed in the
30422 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the
30423 difficulty that the state of process-local resources is usually held in process memory. If a thread
30424 that is not calling *fork()* holds a resource, that resource is never released in the child process
30425 because the thread whose job it is to release the resource does not exist in the child process.

30426 When a programmer is writing a multi-threaded program, the first described use of *fork()*,
30427 creating new threads in the same program, is provided by the *pthread_create()* function. The
30428 *fork()* function is thus used only to run new programs, and the effects of calling functions that
30429 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

30430 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*
30431 function lets all the threads in the parent be duplicated in the child. This essentially duplicates
30432 the state of the parent in the child. This allows threads in the child to continue processing and
30433 allows locks and the state to be preserved without explicit *pthread_atfork()* code. The calling
30434 process has to ensure that the threads processing state that is shared between the parent and
30435 child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall()*. For
30436 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two
30437 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.
30438 If this is not desired behavior, the parent process has to synchronize with such threads before
30439 calling *forkall()*.

30440 While the *fork()* function is async-signal-safe, there is no way for an implementation to
30441 determine whether the fork handlers established by *pthread_atfork()* are async-signal-safe. The
30442 fork handlers may attempt to execute portions of the implementation that are not async-signal-
30443 safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore
30444 undefined for the fork handlers to execute functions that are not async-signal-safe when *fork()*
30445 is called from a signal handler.

30446 When *forkall()* is called, threads, other than the calling thread, that are in functions that can
30447 return with an [EINTR] error may have those functions return [EINTR] if the implementation
30448 cannot ensure that the function behaves correctly in the parent and child. In particular,
30449 *pthread_cond_wait()* and *pthread_cond_timedwait()* need to return in order to ensure that the
30450 condition has not changed. These functions can be awakened by a spurious condition wakeup
30451 rather than returning [EINTR].

30452 **FUTURE DIRECTIONS**

30453 None.

30454 **SEE ALSO**30455 *alarm()*, *exec*, *fcntl()*, *posix_trace_attr_getinherited()*, *posix_trace_eventid_equal()*, *pthread_atfork()*,
30456 *semop()*, *signal()*, *times()*30457 XBD Section 4.12 (on page 111), [<sys/types.h>](#), [<unistd.h>](#)30458 **CHANGE HISTORY**

30459 First released in Issue 1. Derived from Issue 1 of the SVID.

30460 **Issue 5**30461 The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX
30462 Threads Extension.30463 **Issue 6**30464 The following new requirements on POSIX implementations derive from alignment with the
30465 Single UNIX Specification:30466 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
30467 required for conforming implementations of previous POSIX specifications, it was not
30468 required for UNIX applications.

30469 The following changes were made to align with the IEEE P1003.1a draft standard:

30470 The effect of *fork()* on a pending alarm call in the child process is clarified.

30471 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

30472 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

30473 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the
30474 DESCRIPTION and RATIONALE relating to fork handlers registered by the *pthread_atfork()*
30475 function and async-signal safety.30476 **Issue 7**30477 Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous
30478 input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.30479 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers,
30480 and Threads options is moved to the Base.

30481 Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

30482 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0123 [858] is applied.

30483 **NAME**

30484 `fpathconf, pathconf` ‡get configurable pathname variables

30485 **SYNOPSIS**

```
30486 #include <unistd.h>
30487 long fpathconf(int fildes, int name);
30488 long pathconf(const char *path, int name);
```

30489 **DESCRIPTION**

30490 The *fpathconf()* and *pathconf()* functions shall determine the current value of a configurable limit
30491 or option (*variable*) that is associated with a file or directory.

30492 For *pathconf()*, the *path* argument points to the pathname of a file or directory.

30493 For *fpathconf()*, the *fildes* argument is an open file descriptor.

30494 The *name* argument represents the variable to be queried relative to that file or directory.
30495 Implementations shall support all of the variables listed in the following table and may support
30496 others. The variables in the following table come from `<limits.h>` or `<unistd.h>` and the
30497 symbolic constants, defined in `<unistd.h>`, are the corresponding values used for *name*.

Variable	Value of <i>name</i>	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	4,7
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3,4
{PATH_MAX}	_PC_PATH_MAX	4,5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX2_SYMLINKS}	_PC_2_SYMLINKS	4
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	10
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	10
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	10
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	10
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	10
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4,9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8
_POSIX_TIMESTAMP_RESOLUTION	_PC_TIMESTAMP_RESOLUTION	1

30520 **Requirements**

- 30521 1. If *path* or *fildes* refers to a directory, the value returned shall apply to the directory itself.
- 30522 2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an
30523 implementation supports an association of the variable name with the specified file.
- 30524 3. If *path* or *fildes* refers to a directory, the value returned shall apply to filenames within the
30525 directory.

- 30526 4. If *path* or *filde*s does not refer to a directory, it is unspecified whether an implementation
30527 supports an association of the variable name with the specified file.
- 30528 5. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of a
30529 relative pathname that would not cross any mount points when the specified directory is
30530 the working directory.
- 30531 6. If *path* refers to a FIFO, or *filde*s refers to a pipe or FIFO, the value returned shall apply to
30532 the referenced object. If *path* or *filde*s refers to a directory, the value returned shall apply to
30533 any FIFO that exists or can be created within the directory. If *path* or *filde*s refers to any
30534 other type of file, it is unspecified whether an implementation supports an association of
30535 the variable name with the specified file.
- 30536 7. If *path* or *filde*s refers to a directory, the value returned shall apply to any files, other than
30537 directories, that exist or can be created within the directory.
- 30538 8. If *path* or *filde*s refers to a directory, it is unspecified whether an implementation supports
30539 an association of the variable name with the specified file.
- 30540 9. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of the
30541 string that a symbolic link in that directory can contain.
- 30542 10. If *path* or *filde*s does not refer to a regular file, it is unspecified whether an
30543 implementation supports an association of the variable name with the specified file. If an
30544 implementation supports such an association for other than a regular file, the value
30545 returned is unspecified.

30546 RETURN VALUE

30547 If *name* is an invalid value, both *pathconf*() and *fpathconf*() shall return -1 and set *errno* to
30548 indicate the error.

30549 If the variable corresponding to *name* is described in **<limits.h>** as a maximum or minimum
30550 value and the variable has no limit for the path or file descriptor, both *pathconf*() and *fpathconf*()
30551 shall return -1 without changing *errno*. Note that indefinite limits do not imply infinite limits;
30552 see **<limits.h>**.

30553 If the implementation needs to use *path* to determine the value of *name* and the implementation
30554 does not support the association of *name* with the file specified by *path*, or if the process did not
30555 have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf*()
30556 shall return -1 and set *errno* to indicate the error.

30557 If the implementation needs to use *filde*s to determine the value of *name* and the implementation
30558 does not support the association of *name* with the file specified by *filde*s, or if *filde*s is an invalid
30559 file descriptor, *fpathconf*() shall return -1 and set *errno* to indicate the error.

30560 Otherwise, *pathconf*() or *fpathconf*() shall return the current variable value for the file or
30561 directory without changing *errno*. The value returned shall not be more restrictive than the
30562 corresponding value available to the application when it was compiled with the
30563 implementation's **<limits.h>** or **<unistd.h>**.

30564 If the variable corresponding to *name* is dependent on an unsupported option, the results are
30565 unspecified.

30566 ERRORS

30567 The *pathconf*() function shall fail if:

30568 [EINVAL] The value of *name* is not valid.

- 30569 [EOVERFLOW] The value of *name* is `_PC_TIMESTAMP_RESOLUTION` and the resolution is
30570 larger than `{LONG_MAX}`.
- 30571 The *pathconf()* function may fail if:
- 30572 [EACCES] Search permission is denied for a component of the path prefix.
- 30573 [EINVAL] The implementation does not support an association of the variable *name* with
30574 the specified file.
- 30575 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
30576 argument.
- 30577 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during
30578 resolution of the *path* argument.
- 30579 [ENAMETOOLONG]
30580 The length of a component of a pathname is longer than `{NAME_MAX}`.
- 30581 [ENAMETOOLONG]
30582 The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a
30583 symbolic link produced an intermediate result with a length that exceeds
30584 `{PATH_MAX}`.
- 30585 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 30586 [ENOTDIR] A component of the path prefix names an existing file that is neither a
30587 directory nor a symbolic link to a directory, or the *path* argument contains at
30588 least one non-`<slash>` character and ends with one or more trailing `<slash>`
30589 characters and the last pathname component names an existing file that is
30590 neither a directory nor a symbolic link to a directory.
- 30591 The *fpathconf()* function shall fail if:
- 30592 [EINVAL] The value of *name* is not valid.
- 30593 [EOVERFLOW] The value of *name* is `_PC_TIMESTAMP_RESOLUTION` and the resolution is
30594 larger than `{LONG_MAX}`.
- 30595 The *fpathconf()* function may fail if:
- 30596 [EBADF] The *fildev* argument is not a valid file descriptor.
- 30597 [EINVAL] The implementation does not support an association of the variable *name* with
30598 the specified file.

EXAMPLES

30599 None.
30600

APPLICATION USAGE

30601 Application developers should check whether an option, such as `_POSIX_ADVISORY_INFO`, is
30602 supported prior to obtaining and using values for related variables such as
30603 `{POSIX_ALLOC_SIZE_MIN}`.
30604

RATIONALE

30605 The *pathconf()* function was proposed immediately after the *sysconf()* function when it was
30606 realized that some configurable values may differ across file system, directory, or device
30607 boundaries.
30608

30609 For example, `{NAME_MAX}` frequently changes between System V and BSD-based file systems;
30610 System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file

30611 systems, an application would be forced to limit all pathname components to 14 bytes, as this
 30612 would be the value specified in `<limits.h>` on such a system.

30613 Therefore, various useful values can be queried on any pathname or file descriptor, assuming
 30614 that appropriate privileges are in place.

30615 The value returned for the variable `{PATH_MAX}` indicates the longest relative pathname that
 30616 could be given if the specified directory is the current working directory of the process. A
 30617 process may not always be able to generate a name that long and use it if a subdirectory in the
 30618 pathname crosses into a more restrictive file system. Note that implementations are allowed to
 30619 accept pathnames longer than `{PATH_MAX}` bytes long, but are not allowed to return
 30620 pathnames longer than this unless the user specifies a larger buffer using a function that
 30621 provides a buffer size argument.

30622 The value returned for the variable `_POSIX_CHOWN_RESTRICTED` also applies to directories
 30623 that do not have file systems mounted on them. The value may change when crossing a mount
 30624 point, so applications that need to know should check for each directory. (An even easier check
 30625 is to try the `chown()` function and look for an error in case it happens.)

30626 Unlike the values returned by `sysconf()`, the pathname-oriented variables are potentially more
 30627 volatile and are not guaranteed to remain constant throughout the lifetime of the process. For
 30628 example, in between two calls to `pathconf()`, the file system in question may have been
 30629 unmounted and remounted with different characteristics.

30630 Also note that most of the errors are optional. If one of the variables always has the same value
 30631 on an implementation, the implementation need not look at `path` or `fildev` to return that value and
 30632 is, therefore, not required to detect any of the errors except the meaning of `[EINVAL]` that
 30633 indicates that the value of `name` is not valid for that variable, and the `[EOVERFLOW]` error that
 30634 indicates the value to be returned is larger than `{LONG_MAX}`.

30635 If the value of any of the limits is unspecified (logically infinite), they will not be defined in
 30636 `<limits.h>` and the `pathconf()` and `fpathconf()` functions return `-1` without changing `errno`. This
 30637 can be distinguished from the case of giving an unrecognized `name` argument because `errno` is set
 30638 to `[EINVAL]` in this case.

30639 Since `-1` is a valid return value for the `pathconf()` and `fpathconf()` functions, applications should
 30640 set `errno` to zero before calling them and check `errno` only if the return value is `-1`.

30641 For the case of `{SYMLINK_MAX}`, since both `pathconf()` and `open()` follow symbolic links, there
 30642 is no way that `path` or `fildev` could refer to a symbolic link.

30643 It was the intention of IEEE Std 1003.1d-1999 that the following variables:

```
30644     {POSIX_ALLOC_SIZE_MIN}
30645     {POSIX_REC_INCR_XFER_SIZE}
30646     {POSIX_REC_MAX_XFER_SIZE}
30647     {POSIX_REC_MIN_XFER_SIZE}
30648     {POSIX_REC_XFER_ALIGN}
```

30649 only applied to regular files, but Note 10 also permits implementation of the advisory semantics
 30650 on other file types unique to an implementation (for example, a character special device).

30651 The `[EOVERFLOW]` error for `_PC_TIMESTAMP_RESOLUTION` cannot occur on POSIX-
 30652 compliant file systems because POSIX requires a timestamp resolution no larger than one
 30653 second. Even on 32-bit systems, this can be represented without overflow.

30654 **FUTURE DIRECTIONS**

30655 None.

30656 **SEE ALSO**30657 *chown()*, *confstr()*, *sysconf()*30658 XBD [<limits.h>](#), [<unistd.h>](#)30659 XCU *getconf*30660 **CHANGE HISTORY**

30661 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

30662 **Issue 5**

30663 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

30664 Large File Summit extensions are added.

30665 **Issue 6**30666 The following new requirements on POSIX implementations derive from alignment with the
30667 Single UNIX Specification:

30668 The DESCRIPTION is updated to include {FILESIZEBITS}.

30669 The [ELOOP] mandatory error condition is added.

30670 A second [ENAMETOOLONG] is added as an optional error condition.

30671 The following changes were made to align with the IEEE P1003.1a draft standard:

30672 The `_PC_SYMLINK_MAX` entry is added to the table in the DESCRIPTION.30673 The following *pathconf()* variables and their associated names are added for alignment with
30674 IEEE Std 1003.1d-1999:

30675 {POSIX_ALLOC_SIZE_MIN}

30676 {POSIX_REC_INCR_XFER_SIZE}

30677 {POSIX_REC_MAX_XFER_SIZE}

30678 {POSIX_REC_MIN_XFER_SIZE}

30679 {POSIX_REC_XFER_ALIGN}

30680 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth
30681 paragraph of the DESCRIPTION and removing shading and margin markers from the table.
30682 This change is needed since implementations are required to support all of these symbols.30683 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/34 is applied, adding the table entry for
30684 POSIX2_SYMLINKS in the DESCRIPTION.30685 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/35 is applied, updating the
30686 DESCRIPTION and RATIONALE sections to clarify behavior for the following variables:

30687 {POSIX_ALLOC_SIZE_MIN}

30688 {POSIX_REC_INCR_XFER_SIZE}

30689 {POSIX_REC_MAX_XFER_SIZE}

30690 {POSIX_REC_MIN_XFER_SIZE}

30691 {POSIX_REC_XFER_ALIGN}

30692 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/36 is applied, updating the RETURN
30693 VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable
30694 is dependent on an unsupported option, and advising application developers to check for

- 30695 supported options prior to obtaining and using such values.
- 30696 **Issue 7**
- 30697 Austin Group Interpretations 1003.1-2001 #143 and #160 are applied.
- 30698 Changes are made related to support for finegrained timestamps.
- 30699 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
30700 pathname exists but is not a directory or a symbolic link to a directory.
- 30701 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0160 [256,428], XSH/TC1-2008/0161
30702 [256,428], and XSH/TC1-2008/0162 [324] are applied.
- 30703 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0124 [651] and XSH/TC2-2008/0125
30704 [651] are applied.

30705 **NAME**

30706 fpclassify — classify real floating type

30707 **SYNOPSIS**

30708 #include <math.h>

30709 int fpclassify(real-floating x);

30710 **DESCRIPTION**

30711 CX The functionality described on this reference page is aligned with the ISO C standard. Any
30712 conflict between the requirements described here and the ISO C standard is unintentional. This
30713 volume of POSIX.1-2017 defers to the ISO C standard.

30714 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,
30715 zero, or into another implementation-defined category. First, an argument represented in a
30716 format wider than its semantic type is converted to its semantic type. Then classification is based
30717 on the type of the argument.

30718 **RETURN VALUE**

30719 The *fpclassify()* macro shall return the value of the number classification macro appropriate to
30720 the value of its argument.

30721 **ERRORS**

30722 No errors are defined.

30723 **EXAMPLES**

30724 None.

30725 **APPLICATION USAGE**

30726 None.

30727 **RATIONALE**

30728 None.

30729 **FUTURE DIRECTIONS**

30730 None.

30731 **SEE ALSO**30732 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

30733 XBD <math.h>

30734 **CHANGE HISTORY**

30735 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

30736 **NAME**30737 `dprintf, fprintf, printf, snprintf, sprintf` ‡'print formatted output30738 **SYNOPSIS**30739 `#include <stdio.h>`

```

30740 CX int dprintf(int filides, const char *restrict format, ...);
30741 int fprintf(FILE *restrict stream, const char *restrict format, ...);
30742 int printf(const char *restrict format, ...);
30743 int snprintf(char *restrict s, size_t n,
30744     const char *restrict format, ...);
30745 int sprintf(char *restrict s, const char *restrict format, ...);

```

30746 **DESCRIPTION**

30747 CX Excluding `dprintf()`: The functionality described on this reference page is aligned with the ISO C
 30748 standard. Any conflict between the requirements described here and the ISO C standard is
 30749 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

30750 The `fprintf()` function shall place output on the named output *stream*. The `printf()` function shall
 30751 place output on the standard output stream *stdout*. The `sprintf()` function shall place output
 30752 followed by the null byte, `'\0'`, in consecutive bytes starting at *s*; it is the user's responsibility
 30753 to ensure that enough space is available.

30754 CX The `dprintf()` function shall be equivalent to the `fprintf()` function, except that `dprintf()` shall
 30755 write output to the file associated with the file descriptor specified by the *filides* argument rather
 30756 than place output on a stream.

30757 The `snprintf()` function shall be equivalent to `sprintf()`, with the addition of the *n* argument
 30758 which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s*
 30759 may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of
 30760 being written to the array, and a null byte is written at the end of the bytes actually written into
 30761 the array.

30762 If copying takes place between objects that overlap as a result of a call to `sprintf()` or `snprintf()`,
 30763 the results are undefined.

30764 Each of these functions converts, formats, and prints its arguments under control of the *format*.
 30765 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is
 30766 composed of zero or more directives: *ordinary characters*, which are simply copied to the output
 30767 stream, and *conversion specifications*, each of which shall result in the fetching of zero or more
 30768 arguments. The results are undefined if there are insufficient arguments for the *format*. If the
 30769 *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are
 30770 otherwise ignored.

30771 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 30772 to the next unused argument. In this case, the conversion specifier character `%` (see below) is
 30773 replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range `[1, {NL_ARGMAX}]`,
 30774 giving the position of the argument in the argument list. This feature provides for the definition
 30775 of format strings that select arguments in an order appropriate to specific languages (see the
 30776 EXAMPLES section).

30777 The *format* can contain either numbered argument conversion specifications (that is, `"%n$"` and
 30778 `"*m$"`), or unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The
 30779 only exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing
 30780 numbered and unnumbered argument specifications in a *format* string are undefined. When
 30781 numbered argument specifications are used, specifying the *N*th argument requires that all the
 30782 leading arguments, from the first to the $(N-1)$ th, are specified in the format string.

30783 In format strings containing the "%n\$" form of conversion specification, numbered arguments
 30784 in the argument list can be referenced from the format string as many times as required.

30785 In format strings containing the % form of conversion specification, each conversion specification
 30786 uses the first unused argument in the argument list.

30787 CX All forms of the fprintf() functions allow for the insertion of a language-dependent radix
 30788 character in the output string. The radix character is defined in the current locale (category
 30789 LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the
 30790 radix character shall default to a <period> ('.').

30791 CX Each conversion specification is introduced by the '%' character or by the character sequence
 30792 "%n\$", after which the following appear in sequence:

30793 Zero or more *flags* (in any order), which modify the meaning of the conversion
 30794 specification.

30795 An optional minimum *field width*. If the converted value has fewer bytes than the field
 30796 width, it shall be padded with <space> characters by default on the left; it shall be padded
 30797 on the right if the left-adjustment flag ('-'), described below, is given to the field width.
 30798 The field width takes the form of an <asterisk> ('*'), described below, or a decimal
 30799 integer.

30800 An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,
 30801 x, and X conversion specifiers; the number of digits to appear after the radix character for
 30802 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for
 30803 the g and G conversion specifiers; or the maximum number of bytes to be printed from a
 30804 XSI string in the s and S conversion specifiers. The precision takes the form of a <period>
 30805 ('.') followed either by an <asterisk> ('*'), described below, or an optional decimal digit
 30806 string, where a null digit string is treated as zero. If a precision appears with any other
 30807 conversion specifier, the behavior is undefined.

30808 An optional length modifier that specifies the size of the argument.

30809 A *conversion specifier* character that indicates the type of conversion to be applied.

30810 A field width, or precision, or both, may be indicated by an <asterisk> ('*'). In this case an
 30811 argument of type **int** supplies the field width or precision. Applications shall ensure that
 30812 arguments specifying field width, or precision, or both appear in that order before the argument,
 30813 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field
 30814 CX width. A negative precision is taken as if the precision were omitted. In *format* strings
 30815 containing the "%n\$" form of a conversion specification, a field width or precision may be
 30816 indicated by the sequence "*m\$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}]
 30817 giving the position in the argument list (after the *format* argument) of an integer argument
 30818 containing the field width or precision, for example:

30819 `printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);`

30820 The flag characters and their meanings are:

30821 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,
 30822 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping characters. For
 30823 other conversions the behavior is undefined. The non-monetary grouping character is
 30824 used.

30825 - The result of the conversion shall be left-justified within the field. The conversion is
 30826 right-justified if this flag is not specified.

- 30827 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The
 30828 conversion shall begin with a sign only when a negative value is converted if this flag is
 30829 not specified.
- 30830 <space> If the first character of a signed conversion is not a sign or if a signed conversion results
 30831 in no characters, a <space> shall be prefixed to the result. This means that if the
 30832 <space> and '+' flags both appear, the <space> flag shall be ignored.
- 30833 # Specifies that the value is to be converted to an alternative form. For o conversion, it
 30834 shall increase the precision, if and only if necessary, to force the first digit of the result
 30835 to be a zero (if the value and precision are both 0, a single 0 is printed). For x or X
 30836 conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e,
 30837 E, f, F, g, and G conversion specifiers, the result shall always contain a radix character,
 30838 even if no digits follow the radix character. Without this flag, a radix character appears
 30839 in the result of these conversions only if a digit follows it. For g and G conversion
 30840 specifiers, trailing zeros shall *not* be removed from the result as they normally are. For
 30841 other conversion specifiers, the behavior is undefined.
- 30842 0 For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros
 30843 (following any indication of sign or base) are used to pad to the field width rather than
 30844 performing space padding, except when converting an infinity or NaN. If the '0' and
 30845 '-' flags both appear, the '0' flag is ignored. For d, i, o, u, x, and X conversion
 30846 CX specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and
 30847 <apostrophe> flags both appear, the grouping characters are inserted before zero
 30848 padding. For other conversions, the behavior is undefined.
- 30849 The length modifiers and their meanings are:
- 30850 hh Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **signed char**
 30851 or **unsigned char** argument (the argument will have been promoted according to the
 30852 integer promotions, but its value shall be converted to **signed char** or **unsigned char**
 30853 before printing); or that a following n conversion specifier applies to a pointer to a
 30854 **signed char** argument.
- 30855 h Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **short** or
 30856 **unsigned short** argument (the argument will have been promoted according to the
 30857 integer promotions, but its value shall be converted to **short** or **unsigned short** before
 30858 printing); or that a following n conversion specifier applies to a pointer to a **short**
 30859 argument.
- 30860 l (ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long** or
 30861 **unsigned long** argument; that a following n conversion specifier applies to a pointer to
 30862 a **long** argument; that a following c conversion specifier applies to a **wint_t** argument;
 30863 that a following s conversion specifier applies to a pointer to a **wchar_t** argument; or
 30864 has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.
- 30865 ll (ell-ell) Specifies that a following d, i, o, u, x, or X conversion specifier applies to a **long long**
 30866 or **unsigned long long** argument; or that a following n conversion specifier applies to a
 30867 pointer to a **long long** argument.
- 30869 j Specifies that a following d, i, o, u, x, or X conversion specifier applies to an **intmax_t**
 30870 or **uintmax_t** argument; or that a following n conversion specifier applies to a pointer
 30871 to an **intmax_t** argument.

30872	z	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a size_t or the corresponding signed integer type argument; or that a following <code>n</code> conversion specifier applies to a pointer to a signed integer type corresponding to a size_t argument.
30873		
30874		
30875	t	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a ptrdiff_t or the corresponding unsigned type argument; or that a following <code>n</code> conversion specifier applies to a pointer to a ptrdiff_t argument.
30876		
30877		
30878	L	Specifies that a following <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , or <code>G</code> conversion specifier applies to a long double argument.
30879		
30880		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
30881		
30882		The conversion specifiers and their meanings are:
30883	d, i	The int argument shall be converted to a signed decimal in the style " <code>[-]dddd</code> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
30884		
30885		
30886		
30887		
30888	o	The unsigned argument shall be converted to unsigned octal format in the style " <code>dddd</code> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
30889		
30890		
30891		
30892		
30893	u	The unsigned argument shall be converted to unsigned decimal format in the style " <code>dddd</code> ". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
30894		
30895		
30896		
30897		
30898	x	The unsigned argument shall be converted to unsigned hexadecimal format in the style " <code>dddd</code> "; the letters "abcdef" are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.
30899		
30900		
30901		
30902		
30903	X	Equivalent to the <code>x</code> conversion specifier, except that letters "ABCDEF" are used instead of "abcdef".
30904		
30905	f, F	The double argument shall be converted to decimal notation in the style " <code>[-]ddd.ddd</code> ", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.
30906		
30907		
30908		
30909		
30910		
30911		A double argument representing an infinity shall be converted in one of the styles " <code>[-]inf</code> " or " <code>[-]infinity</code> "; which style is implementation-defined. A double argument representing a NaN shall be converted in one of the styles " <code>[-]nan(<i>n-char-sequence</i>)</code> " or " <code>[-]nan</code> "; which style, and the meaning of any <i>n-char-sequence</i> , is implementation-defined. The <code>F</code> conversion specifier produces "INF", "INFINITY", or "NAN" instead of "inf", "infinity", or "nan", respectively.
30912		
30913		
30914		
30915		
30916		

30917 e, E The **double** argument shall be converted in the style " $[-]d.ddde\pm dd$ ", where there is
30918 one digit before the radix character (which is non-zero if the argument is non-zero) and
30919 the number of digits after it is equal to the precision; if the precision is missing, it shall
30920 be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall
30921 appear. The low-order digit shall be rounded in an implementation-defined manner.
30922 The E conversion specifier shall produce a number with 'E' instead of 'e'
30923 introducing the exponent. The exponent shall always contain at least two digits. If the
30924 value is zero, the exponent shall be zero.

30925 A **double** argument representing an infinity or NaN shall be converted in the style of
30926 an f or F conversion specifier.

30927 g, G The **double** argument representing a floating-point number shall be converted in the
30928 style f or e (or in the style F or E in the case of a G conversion specifier), depending on
30929 the value converted and the precision. Let P equal the precision if non-zero, 6 if the
30930 precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E
30931 would have an exponent of X:

30932 \nmid if $P \geq X \geq -4$, the conversion shall be with style f (or F) and precision $P - (X + 1)$.

30933 \nmid otherwise, the conversion shall be with style e (or E) and precision $P - 1$.

30934 Finally, unless the '#' flag is used, any trailing zeros shall be removed from the
30935 fractional portion of the result and the decimal-point character shall be removed if there
30936 is no fractional portion remaining.

30937 A **double** argument representing an infinity or NaN shall be converted in the style of
30938 an f or F conversion specifier.

30939 a, A A **double** argument representing a floating-point number shall be converted in the
30940 style " $[-]0xh.hhhhp\pm d$ ", where there is one hexadecimal digit (which shall be non-
30941 zero if the argument is a normalized floating-point number and is otherwise
30942 unspecified) before the decimal-point character and the number of hexadecimal digits
30943 after it is equal to the precision; if the precision is missing and FLT_RADIX is a power
30944 of 2, then the precision shall be sufficient for an exact representation of the value; if the
30945 precision is missing and FLT_RADIX is not a power of 2, then the precision shall be
30946 sufficient to distinguish values of type **double**, except that trailing zeros may be
30947 omitted; if the precision is zero and the '#' flag is not specified, no decimal-point
30948 character shall appear. The letters "abcdef" shall be used for a conversion and the
30949 letters "ABCDEF" for A conversion. The A conversion specifier produces a number with
30950 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one
30951 digit, and only as many more digits as necessary to represent the decimal exponent of
30952 2. If the value is zero, the exponent shall be zero.

30953 A **double** argument representing an infinity or NaN shall be converted in the style of
30954 an f or F conversion specifier.

30955 c The **int** argument shall be converted to an **unsigned char**, and the resulting byte shall
30956 be written.

30957 If an l (ell) qualifier is present, the **wint_t** argument shall be converted as if by an ls
30958 conversion specification with no precision and an argument that points to a two-
30959 element array of type **wchar_t**, the first element of which contains the **wint_t** argument
30960 to the ls conversion specification and the second element contains a null wide
30961 character.

30962	s	The argument shall be a pointer to an array of char . Bytes from the array shall be written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the application shall ensure that the array contains a null byte.
30963		
30964		
30965		
30966		
30967		If an l (ell) qualifier is present, the argument shall be a pointer to an array of type wchar_t . Wide characters from the array shall be converted to characters (each as if by a call to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written.
30968		
30969		
30970		
30971		
30972		
30973		
30974		
30975		
30976		
30977		
30978		
30979	p	The argument shall be a pointer to void . The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.
30980		
30981	n	The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the <i>fprintf()</i> functions. No argument is converted.
30982		
30983		
30984	XSI	C Equivalent to lc .
30985	XSI	S Equivalent to ls .
30986	%	Print a '%' character; no argument is converted. The complete conversion specification shall be %%. If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.
30987		
30988		
30989		
30990		
30991		In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <i>fprintf()</i> and <i>printf()</i> are printed as if <i>fputc()</i> had been called.
30992		
30993		
30994		For the a and A conversion specifiers, if FLT_RADIX is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.
30995		
30996		For a and A conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.
30997		
30998		
30999		
31000		For the e , E , f , F , g , and G conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG , then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L \leq D \leq U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.
31001		
31002		
31003		
31004		
31005		
31006		
31007		

- 31008 CX The last data modification and last file status change timestamps of the file shall be marked for
31009 update:
- 31010 1. Between the call to a successful execution of `fprintf()` or `printf()` and the next successful
31011 completion of a call to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`
 - 31012 2. Upon successful completion of a call to `dprintf()`

31013 RETURN VALUE

- 31014 CX Upon successful completion, the `dprintf()`, `fprintf()`, and `printf()` functions shall return the
31015 number of bytes transmitted.
- 31016 Upon successful completion, the `sprintf()` function shall return the number of bytes written to `s`,
31017 excluding the terminating null byte.
- 31018 Upon successful completion, the `snprintf()` function shall return the number of bytes that would
31019 be written to `s` had `n` been sufficiently large excluding the terminating null byte.
- 31020 CX If an output error was encountered, these functions shall return a negative value and set `errno` to
31021 indicate the error.
- 31022 If the value of `n` is zero on a call to `snprintf()`, nothing shall be written, the number of bytes that
31023 would have been written had `n` been sufficiently large excluding the terminating null shall be
31024 returned, and `s` may be a null pointer.

31025 ERRORS

- 31026 CX For the conditions under which `dprintf()`, `fprintf()`, and `printf()` fail and may fail, refer to `fputc()`
31027 or `fputwc()`.
- 31028 In addition, all forms of `fprintf()` shall fail if:
- 31029 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been
31030 detected.
- 31031 CX [EOVERFLOW] The value to be returned is greater than {INT_MAX}.
- 31032 The `dprintf()` function may fail if:
- 31033 [EBADF] The `files` argument is not a valid file descriptor.
- 31034 CX The `dprintf()`, `fprintf()`, and `printf()` functions may fail if:
- 31035 CX [ENOMEM] Insufficient storage space is available.
- 31036 The `snprintf()` function shall fail if:
- 31037 CX [EOVERFLOW] The value of `n` is greater than {INT_MAX}.

31038 EXAMPLES**31039 Printing Language-Independent Date and Time**

31040 The following statement can be used to print date and time using a language-independent
31041 format:

```
31042 printf(format, weekday, month, day, hour, min);
```

31043 For American usage, `format` could be a pointer to the following string:

```
31044 "%s, %s %d, %d:%.2d\n"
```

31045 This example would produce the following message:

```

31046      Sunday, July 3, 10:02
31047      For German usage, format could be a pointer to the following string:
31048      "%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
31049      This definition of format would produce the following message:
31050      Sonntag, 3. Juli, 10:02

```

Printing File Information

The following example prints information about the type, permissions, and number of links of a specific file in a directory.

The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined *strperm()* function shall return a string similar to the one at the beginning of the output for the following command:

```
31057      ls -l
```

The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()* function shall return a **passwd** structure from which the name of the user is extracted. If the user name is not found, the program instead prints out the numeric value of the user ID.

The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar to *getpwuid()* except that it shall return group information based on the group number. Once again, if the group is not found, the program prints the numeric value of the group for the entry.

The final call to *printf()* prints the size of the file.

```

31065      #include <stdio.h>
31066      #include <sys/types.h>
31067      #include <pwd.h>
31068      #include <grp.h>
31069
31070      char *strperm (mode_t);
31071      ...
31072      struct stat statbuf;
31073      struct passwd *pwd;
31074      struct group *grp;
31075      ...
31076      printf("%10.10s", strperm (statbuf.st_mode));
31077      printf("%4d", statbuf.st_nlink);
31078
31079      if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
31080          printf(" %-8.8s", pwd->pw_name);
31081      else
31082          printf(" %-8ld", (long) statbuf.st_uid);
31083
31084      if ((grp = getgrgid(statbuf.st_gid)) != NULL)
31085          printf(" %-8.8s", grp->gr_name);
31086      else
31087          printf(" %-8ld", (long) statbuf.st_gid);
31088
31089      printf("%9jd", (intmax_t) statbuf.st_size);
31090      ...

```

31087 **Printing a Localized Date String**

31088 The following example gets a localized date string. The `nl_langinfo()` function shall return the
 31089 localized date string, which specifies the order and layout of the date. The `strftime()` function
 31090 takes this information and, using the `tm` structure for values, places the date and time
 31091 information into `datestring`. The `printf()` function then outputs `datestring` and the name of the
 31092 entry.

```
31093 #include <stdio.h>
31094 #include <time.h>
31095 #include <langinfo.h>
31096 ...
31097 struct dirent *dp;
31098 struct tm *tm;
31099 char datestring[256];
31100 ...
31101 strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
31102 printf(" %s %s\n", datestring, dp->d_name);
31103 ...
```

31104 **Printing Error Information**

31105 The following example uses `fprintf()` to write error information to standard error.

31106 In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If
 31107 the file already exists, this is an error, as indicated by the `O_EXCL` flag on the `open()` function. If
 31108 the call fails, the program assumes that someone else is updating the password file, and the
 31109 program exits.

31110 The next group of calls saves a new password file as the current password file by creating a link
 31111 between **LOCKFILE** and the new password file **PASSWDFILE**.

```
31112 #include <sys/types.h>
31113 #include <sys/stat.h>
31114 #include <fcntl.h>
31115 #include <stdio.h>
31116 #include <stdlib.h>
31117 #include <unistd.h>
31118 #include <string.h>
31119 #include <errno.h>
31120 #define LOCKFILE "/etc/ptmp"
31121 #define PASSWDFILE "/etc/passwd"
31122 ...
31123 int pfd;
31124 ...
31125 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
31126     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
31127 {
31128     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
31129     exit(1);
31130 }
31131 ...
31132 if (link(LOCKFILE,PASSWDFILE) == -1) {
```

```

31133         fprintf(stderr, "Link error: %s\n", strerror(errno));
31134         exit(1);
31135     }
31136     ...

```

31137 **Printing Usage Information**

31138 The following example checks to make sure the program has the necessary arguments, and uses
31139 *fprintf()* to print usage information if the expected number of arguments is not present.

```

31140     #include <stdio.h>
31141     #include <stdlib.h>
31142     ...
31143     char *Options = "hdbt1";
31144     ...
31145     if (argc < 2) {
31146         fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
31147     }
31148     ...

```

31149 **Formatting a Decimal String**

31150 The following example prints a key and data pair on *stdout*. Note use of the <asterisk> ('*') in
31151 the format string; this ensures the correct number of decimal places for the element based on the
31152 number of elements requested.

```

31153     #include <stdio.h>
31154     ...
31155     long i;
31156     char *keyststr;
31157     int elementlen, len;
31158     ...
31159     while (len < elementlen) {
31160         ...
31161         printf("%s Element%0*ld\n", keyststr, elementlen, i);
31162         ...
31163     }

```

31164 **Creating a Pathname**

31165 The following example creates a pathname using information from a previous *getpwnam()*
31166 function that returned the password database entry of the user.

```

31167     #include <stdint.h>
31168     #include <stdio.h>
31169     #include <stdlib.h>
31170     #include <string.h>
31171     #include <sys/types.h>
31172     #include <unistd.h>
31173     ...
31174     char *pathname;
31175     struct passwd *pw;
31176     size_t len;

```

```

31177     ...
31178     // digits required for pid_t is number of bits times
31179     // log2(10) = approx 10/33
31180     len = strlen(pw->pw_dir) + 1 + 1+(sizeof(pid_t)*80+32)/33 +
31181         sizeof ".out";
31182     pathname = malloc(len);
31183     if (pathname != NULL)
31184     {
31185         snprintf(pathname, len, "%s/%jd.out", pw->pw_dir,
31186             (intmax_t) getpid());
31187         ...
31188     }

```

31189 Reporting an Event

31190 The following example loops until an event has timed out. The *pause()* function waits forever
31191 unless it receives a signal. The *fprintf()* statement should never occur due to the possible return
31192 values of *pause()*.

```

31193     #include <stdio.h>
31194     #include <unistd.h>
31195     #include <string.h>
31196     #include <errno.h>
31197     ...
31198     while (!event_complete) {
31199         ...
31200         if (pause() != -1 || errno != EINTR)
31201             fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
31202     }
31203     ...

```

31204 Printing Monetary Information

31205 The following example uses *strfmon()* to convert a number and store it as a formatted monetary
31206 string named *convbuf*. If the first number is printed, the program prints the format and the
31207 description; otherwise, it just prints the number.

```

31208     #include <monetary.h>
31209     #include <stdio.h>
31210     ...
31211     struct tblfmt {
31212         char *format;
31213         char *description;
31214     };
31215     struct tblfmt table[] = {
31216         { "%n", "default formatting" },
31217         { "%11n", "right align within an 11 character field" },
31218         { "%#5n", "aligned columns for values up to 99999" },
31219         { "%=#5n", "specify a fill character" },
31220         { "%=0#5n", "fill characters do not use grouping" },
31221         { "%^#5n", "disable the grouping separator" },
31222         { "%^#5.0n", "round off to whole units" },

```

```

31223         { "%^#5.4n", "increase the precision" },
31224         { "%(#5n", "use an alternative pos/neg style" },
31225         { "%!(#5n", "disable the currency symbol" },
31226     };
31227     ...
31228     float input[3];
31229     int i, j;
31230     char convbuf[100];
31231     ...
31232     strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);

31233     if (j == 0) {
31234         printf("%s%s%s\n", table[i].format,
31235             convbuf, table[i].description);
31236     }
31237     else {
31238         printf("%s\n", convbuf);
31239     }
31240     ...

```

31241 Printing Wide Characters

31242 The following example prints a series of wide characters. Suppose that "L`@`" expands to three
31243 bytes:

```

31244     wchar_t wz [3] = L"@@";           // Zero-terminated
31245     wchar_t wn [3] = L"@@";           // Unterminated

31246     fprintf (stdout, "%ls", wz);      // Outputs 6 bytes
31247     fprintf (stdout, "%ls", wn);      // Undefined because wn has no terminator
31248     fprintf (stdout, "%4ls", wz);     // Outputs 3 bytes
31249     fprintf (stdout, "%4ls", wn);     // Outputs 3 bytes; no terminator needed
31250     fprintf (stdout, "%9ls", wz);     // Outputs 6 bytes
31251     fprintf (stdout, "%9ls", wn);     // Outputs 9 bytes; no terminator needed
31252     fprintf (stdout, "%10ls", wz);    // Outputs 6 bytes
31253     fprintf (stdout, "%10ls", wn);    // Undefined because wn has no terminator

```

31254 In the last line of the example, after processing three characters, nine bytes have been output.
31255 The fourth character must then be examined to determine whether it converts to one byte or
31256 more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth
31257 character in the array, the behavior is undefined.

31258 APPLICATION USAGE

31259 If the application calling *fprintf()* has any objects of type **wint_t** or **wchar_t**, it must also include
31260 the **<wchar.h>** header to have these objects defined.

31261 RATIONALE

31262 If an implementation detects that there are insufficient arguments for the format, it is
31263 recommended that the function should fail and report an [EINVAL] error.

31264 FUTURE DIRECTIONS

31265 None.

31266 **SEE ALSO**31267 [Section 2.5](#) (on page 495), [fprintf\(\)](#), [fscanf\(\)](#), [setlocale\(\)](#), [strfmon\(\)](#), [wctomb\(\)](#)31268 [XBD Chapter 7](#) (on page 135), [<inttypes.h>](#), [<stdio.h>](#), [<wchar.h>](#)31269 **CHANGE HISTORY**

31270 First released in Issue 1. Derived from Issue 1 of the SVID.

31271 **Issue 5**31272 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier can
31273 now be used with `c` and `s` conversion specifiers.31274 The `snprintf()` function is new in Issue 5.31275 **Issue 6**

31276 Extensions beyond the ISO C standard are marked.

31277 The normative text is updated to avoid use of the term “must” for application requirements.

31278 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

31279 The prototypes for `fprintf()`, `printf()`, `snprintf()`, and `sprintf()` are updated, and the XSI
31280 shading is removed from `snprintf()`.31281 The description of `snprintf()` is aligned with the ISO C standard. Note that this supersedes
31282 the `snprintf()` description in The Open Group Base Resolution bwg98-006, which changed
31283 the behavior from Issue 5.

31284 The DESCRIPTION is updated.

31285 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
31286 specification” consistently.

31287 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

31288 An example of printing wide characters is added.

31289 **Issue 7**31290 Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the `0`
31291 flag.

31292 Austin Group Interpretation 1003.1-2001 #170 is applied.

31293 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied,
31294 revising the description of `g` and `G`.

31295 SD5-XSH-ERN-174 is applied.

31296 The `dprintf()` function is added from The Open Group Technical Standard, 2006, Extended API
31297 Set Part 1.31298 Functionality relating to the `%n$` form of conversion specification and the `<apostrophe>` flag is
31299 moved from the XSI option to the Base.

31300 Changes are made related to support for finegrained timestamps.

31301 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0163 [302], XSH/TC1-2008/0164 [316],
31302 XSH/TC1-2008/0165 [316], XSH/TC1-2008/0166 [451,291], and XSH/TC1-2008/0167 [14] are
31303 applied.

31304
31305

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0126 [894], XSH/TC2-2008/0127 [557], and XSH/TC2-2008/0128 [936] are applied.

31306 **NAME**

31307 fputc — put a byte on a stream

31308 **SYNOPSIS**

31309 #include <stdio.h>

31310 int fputc(int *c*, FILE **stream*);31311 **DESCRIPTION**31312 CX The functionality described on this reference page is aligned with the ISO C standard. Any
31313 conflict between the requirements described here and the ISO C standard is unintentional. This
31314 volume of POSIX.1-2017 defers to the ISO C standard.31315 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the
31316 output stream pointed to by *stream*, at the position indicated by the associated file-position
31317 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file
31318 cannot support positioning requests, or if the stream was opened with append mode, the byte
31319 shall be appended to the output stream.31320 CX The last data modification and last file status change timestamps of the file shall be marked for
31321 update between the successful execution of *fputc()* and the next successful completion of a call
31322 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.31323 **RETURN VALUE**31324 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall
31325 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the
31326 error.31327 **ERRORS**31328 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream's* buffer needs to be
31329 flushed, and:31330 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
31331 the thread would be delayed in the write operation.31332 CX [EBADF] The file descriptor underlying *stream* is not a valid file descriptor open for
31333 writing.

31334 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size.

31335 XSI [EFBIG] An attempt was made to write to a file that exceeds the file size limit of the
31336 process.31337 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
31338 offset maximum.31339 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
31340 was transferred.31341 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
31342 process group attempting to write to its controlling terminal, TOSTOP is set,
31343 the calling thread is not blocking SIGTTOU, the process is not ignoring
31344 SIGTTOU, and the process group of the process is orphaned. This error may
31345 also be returned under implementation-defined conditions.

31346 CX [ENOSPC] There was no free space remaining on the device containing the file.

31347 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
31348 any process. A SIGPIPE signal shall also be sent to the thread.

- 31349 The *fputc()* function may fail if:
- 31350 CX [ENOMEM] Insufficient storage space is available.
- 31351 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
31352 capabilities of the device.
- 31353 **EXAMPLES**
- 31354 None.
- 31355 **APPLICATION USAGE**
- 31356 None.
- 31357 **RATIONALE**
- 31358 None.
- 31359 **FUTURE DIRECTIONS**
- 31360 None.
- 31361 **SEE ALSO**
- 31362 [Section 2.5](#) (on page 495), *ferror()*, *fopen()*, *getrlimit()*, *putc()*, *puts()*, *setbuf()*, *ulimit()*
- 31363 XBD [<stdio.h>](#)
- 31364 **CHANGE HISTORY**
- 31365 First released in Issue 1. Derived from Issue 1 of the SVID.
- 31366 **Issue 5**
- 31367 Large File Summit extensions are added.
- 31368 **Issue 6**
- 31369 Extensions beyond the ISO C standard are marked.
- 31370 The following new requirements on POSIX implementations derive from alignment with the
31371 Single UNIX Specification:
- 31372 The [EIO] and [EFBIG] mandatory error conditions are added.
- 31373 The [ENOMEM] and [ENXIO] optional error conditions are added.
- 31374 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/37 is applied, updating the [EAGAIN]
31375 error in the ERRORS section from “the process would be delayed” to “the thread would be
31376 delayed”.
- 31377 **Issue 7**
- 31378 Changes are made related to support for finegrained timestamps.
- 31379 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0168 [79] and XSH/TC1-2008/0169
31380 [14] are applied.

31381 **NAME**

31382 fputs — put a string on a stream

31383 **SYNOPSIS**

31384 #include <stdio.h>

31385 int fputs(const char *restrict s, FILE *restrict stream);

31386 **DESCRIPTION**

31387 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 31388 conflict between the requirements described here and the ISO C standard is unintentional. This
 31389 volume of POSIX.1-2017 defers to the ISO C standard.

31390 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed
 31391 to by *stream*. The terminating null byte shall not be written.

31392 CX The last data modification and last file status change timestamps of the file shall be marked for
 31393 update between the successful execution of *fputs()* and the next successful completion of a call
 31394 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

31395 **RETURN VALUE**

31396 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall
 31397 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

31398 **ERRORS**31399 Refer to *fputc()*.31400 **EXAMPLES**31401 **Printing to Standard Output**

31402 The following example gets the current time, converts it to a string using *localtime()* and
 31403 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
 31404 an event for which it is waiting.

```
31405 #include <time.h>
31406 #include <stdio.h>
31407 ...
31408 time_t now;
31409 int minutes_to_event;
31410 ...
31411 time(&now);
31412 printf("The time is ");
31413 fputs(asctime(localtime(&now)), stdout);
31414 printf("There are still %d minutes to the event.\n",
31415     minutes_to_event);
31416 ...
```

31417 **APPLICATION USAGE**31418 The *puts()* function appends a <newline> while *fputs()* does not.

31419 This volume of POSIX.1-2017 requires that successful completion simply return a non-negative
 31420 integer. There are at least three known different implementation conventions for this
 31421 requirement:

31422 Return a constant value.

- 31423 Return the last character written.
- 31424 Return the number of bytes written. Note that this implementation convention cannot be
31425 adhered to for strings longer than {INT_MAX} bytes as the value would not be
31426 representable in the return type of the function. For backwards-compatibility,
31427 implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and
31428 return {INT_MAX} for all longer strings.
- 31429 **RATIONALE**
- 31430 The *fputs()* function is one whose source code was specified in the referenced *The C Programming*
31431 *Language*. In the original edition, the function had no defined return value, yet many practical
31432 implementations would, as a side-effect, return the value of the last character written as that was
31433 the value remaining in the accumulator used as a return value. In the second edition of the book,
31434 either the fixed value 0 or EOF would be returned depending upon the return value of *feof()*;
31435 however, for compatibility with extant implementations, several implementations would, upon
31436 success, return a positive value representing the last byte written.
- 31437 **FUTURE DIRECTIONS**
- 31438 None.
- 31439 **SEE ALSO**
- 31440 [Section 2.5](#) (on page 495), [fopen\(\)](#), [putc\(\)](#), [puts\(\)](#)
- 31441 XBD [<stdio.h>](#)
- 31442 **CHANGE HISTORY**
- 31443 First released in Issue 1. Derived from Issue 1 of the SVID.
- 31444 **Issue 6**
- 31445 Extensions beyond the ISO C standard are marked.
- 31446 The *fputs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.
- 31447 **Issue 7**
- 31448 Changes are made related to support for finegrained timestamps.
- 31449 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0170 [174,412], XSH/TC1-2008/0171
31450 [412], and XSH/TC1-2008/0172 [14] are applied.

31451 **NAME**31452 `fputc` — put a wide-character code on a stream31453 **SYNOPSIS**31454 `#include <stdio.h>`31455 `#include <wchar.h>`31456 `wint_t fputc(wchar_t wc, FILE *stream);`31457 **DESCRIPTION**31458 CX The functionality described on this reference page is aligned with the ISO C standard. Any
31459 conflict between the requirements described here and the ISO C standard is unintentional. This
31460 volume of POSIX.1-2017 defers to the ISO C standard.31461 The `fputc()` function shall write the character corresponding to the wide-character code `wc` to
31462 the output stream pointed to by `stream`, at the position indicated by the associated file-position
31463 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot
31464 support positioning requests, or if the stream was opened with append mode, the character is
31465 appended to the output stream. If an error occurs while writing the character, the shift state of
31466 the output file is left in an undefined state.31467 CX The last data modification and last file status change timestamps of the file shall be marked for
31468 update between the successful execution of `fputc()` and the next successful completion of a call
31469 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.31470 The `fputc()` function shall not change the setting of `errno` if successful.31471 **RETURN VALUE**31472 Upon successful completion, `fputc()` shall return `wc`. Otherwise, it shall return WEOF, the error
31473 CX indicator for the stream shall be set, and `errno` shall be set to indicate the error.31474 **ERRORS**31475 The `fputc()` function shall fail if either the stream is unbuffered or data in the `stream`'s buffer
31476 needs to be written, and:31477 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying `stream` and
31478 the thread would be delayed in the write operation.31479 CX [EBADF] The file descriptor underlying `stream` is not a valid file descriptor open for
31480 writing.31481 CX [EFBIG] An attempt was made to write to a file that exceeds the maximum file size or
31482 the file size limit of the process.31483 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
31484 offset maximum associated with the corresponding stream.31485 [EILSEQ] The wide-character code `wc` does not correspond to a valid character.31486 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
31487 was transferred.31488 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
31489 process group attempting to write to its controlling terminal, TOSTOP is set,
31490 the calling thread is not blocking SIGTTOU, the process is not ignoring
31491 SIGTTOU, and the process group of the process is orphaned. This error may
31492 also be returned under implementation-defined conditions.

- 31493 CX [ENOSPC] There was no free space remaining on the device containing the file.
- 31494 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
31495 any process. A SIGPIPE signal shall also be sent to the thread.
- 31496 The *fputc()* function may fail if:
- 31497 CX [ENOMEM] Insufficient storage space is available.
- 31498 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
31499 capabilities of the device.

31500 EXAMPLES

31501 None.

31502 APPLICATION USAGE

31503 None.

31504 RATIONALE

31505 None.

31506 FUTURE DIRECTIONS

31507 None.

31508 SEE ALSO

31509 [Section 2.5](#) (on page 495), *ferror()*, *fopen()*, *setbuf()*, *ulimit()*

31510 XBD [<stdio.h>](#), [<wchar.h>](#)

31511 CHANGE HISTORY

31512 First released in Issue 4. Derived from the MSE working draft.

31513 Issue 5

31514 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
31515 is changed from **wint_t** to **wchar_t**.

31516 The Optional Header (OH) marking is removed from **<stdio.h>**.

31517 Large File Summit extensions are added.

31518 Issue 6

31519 Extensions beyond the ISO C standard are marked.

31520 The following new requirements on POSIX implementations derive from alignment with the
31521 Single UNIX Specification:

31522 The [EFBIG] and [EIO] mandatory error conditions are added.

31523 The [ENOMEM] and [ENXIO] optional error conditions are added.

31524 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/38 is applied, updating the [EAGAIN]
31525 error in the ERRORS section from “the process would be delayed” to “the thread would be
31526 delayed”.

31527 Issue 7

31528 Changes are made related to support for finegrained timestamps.

31529 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0173 [105], XSH/TC1-2008/0174 [79],
31530 and XSH/TC1-2008/0175 [14] are applied.

31531 **NAME**31532 `fputws` — put a wide-character string on a stream31533 **SYNOPSIS**31534 `#include <stdio.h>`31535 `#include <wchar.h>`31536 `int fputws(const wchar_t *restrict ws, FILE *restrict stream);`31537 **DESCRIPTION**31538 CX The functionality described on this reference page is aligned with the ISO C standard. Any
31539 conflict between the requirements described here and the ISO C standard is unintentional. This
31540 volume of POSIX.1-2017 defers to the ISO C standard.31541 The `fputws()` function shall write a character string corresponding to the (null-terminated) wide-
31542 character string pointed to by `ws` to the stream pointed to by `stream`. No character corresponding
31543 to the terminating null wide-character code shall be written.31544 CX The last data modification and last file status change timestamps of the file shall be marked for
31545 update between the successful execution of `fputws()` and the next successful completion of a call
31546 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.31547 **RETURN VALUE**31548 Upon successful completion, `fputws()` shall return a non-negative number. Otherwise, it shall
31549 CX return `-1`, set an error indicator for the stream, and set `errno` to indicate the error.31550 **ERRORS**31551 Refer to `fputwc()`.31552 **EXAMPLES**

31553 None.

31554 **APPLICATION USAGE**31555 The `fputws()` function does not append a `<newline>`.31556 This volume of POSIX.1-2017 requires that successful completion simply return a non-negative
31557 integer. There are at least three known different implementation conventions for this
31558 requirement:

31559 Return a constant value.

31560 Return the last character written.

31561 Return the number of bytes written. Note that this implementation convention cannot be
31562 adhered to for strings longer than `{INT_MAX}` bytes as the value would not be
31563 representable in the return type of the function. For backwards-compatibility,
31564 implementations can return the number of bytes for strings of up to `{INT_MAX}` bytes, and
31565 return `{INT_MAX}` for all longer strings.31566 **RATIONALE**

31567 None.

31568 **FUTURE DIRECTIONS**

31569 None.

31570 **SEE ALSO**31571 [Section 2.5](#) (on page 495), `fopen()`31572 XBD `<stdio.h>`, `<wchar.h>`

31573 **CHANGE HISTORY**

31574 First released in Issue 4. Derived from the MSE working draft.

31575 **Issue 5**

31576 The Optional Header (OH) marking is removed from `<stdio.h>`.

31577 **Issue 6**

31578 Extensions beyond the ISO C standard are marked.

31579 The `fputws()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

31580 **Issue 7**

31581 Changes are made related to support for finegrained timestamps.

31582 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0176 [412] and XSH/TC1-2008/0177
31583 [14] are applied.

31584 **NAME**

31585 fread — binary input

31586 **SYNOPSIS**

```
31587 #include <stdio.h>
31588 size_t fread(void *restrict ptr, size_t size, size_t nitems,
31589 FILE *restrict stream);
```

31590 **DESCRIPTION**

31591 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 31592 conflict between the requirements described here and the ISO C standard is unintentional. This
 31593 volume of POSIX.1-2017 defers to the ISO C standard.

31594 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size
 31595 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall
 31596 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**
 31597 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be
 31598 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the
 31599 file position indicator for the stream is unspecified. If a partial element is read, its value is
 31600 unspecified.

31601 CX The *fread()* function may mark the last data access timestamp of the file associated with *stream*
 31602 for update. The last data access timestamp shall be marked for update by the first successful
 31603 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or
 31604 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

31605 **RETURN VALUE**

31606 Upon successful completion, *fread()* shall return the number of elements successfully read which
 31607 is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()*
 31608 shall return 0 and the contents of the array and the state of the stream remain unchanged.
 31609 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall
 31610 be set to indicate the error.

31611 **ERRORS**31612 Refer to *fgetc()*.31613 **EXAMPLES**31614 **Reading from a Stream**

31615 The following example transfers a single 100-byte fixed length record from the *fp* stream into the
 31616 array pointed to by *buf*.

```
31617 #include <stdio.h>
31618 ...
31619 size_t elements_read;
31620 char buf[100];
31621 FILE *fp;
31622 ...
31623 elements_read = fread(buf, sizeof(buf), 1, fp);
31624 ...
```

31625 If a read error occurs, *elements_read* will be zero but the number of bytes read from the stream
 31626 could be anything from zero to *sizeof(buf)*-1.

31627 The following example reads multiple single-byte elements from the *fp* stream into the array
 31628 pointed to by *buf*.

```

31629     #include <stdio.h>
31630     ...
31631     size_t bytes_read;
31632     char buf[100];
31633     FILE *fp;
31634     ...
31635     bytes_read = fread(buf, 1, sizeof(buf), fp);
31636     ...

```

31637 If a read error occurs, *bytes_read* will contain the number of bytes read from the stream.

31638 APPLICATION USAGE

31639 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
 31640 end-of-file condition.

31641 Because of possible differences in element length and byte ordering, files written using *fwrite()*
 31642 are application-dependent, and possibly cannot be read using *fread()* by a different application
 31643 or by the same application on a different processor.

31644 RATIONALE

31645 None.

31646 FUTURE DIRECTIONS

31647 None.

31648 SEE ALSO

31649 [Section 2.5](#) (on page 495), *feof()*, *ferror()*, *fgetc()*, *fopen()*, *fscanf()*, *getc()*, *gets()*

31650 XBD [<stdio.h>](#)

31651 CHANGE HISTORY

31652 First released in Issue 1. Derived from Issue 1 of the SVID.

31653 Issue 6

31654 Extensions beyond the ISO C standard are marked.

31655 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

31656 The *fread()* prototype is updated.

31657 The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

31658 Issue 7

31659 Changes are made related to support for finegrained timestamps.

31660 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0178 [232] and XSH/TC1-2008/0179
 31661 [14] are applied.

31662 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0129 [926] is applied.

31663 **NAME**

31664 free — free allocated memory

31665 **SYNOPSIS**

31666 #include <stdlib.h>

31667 void free(void *ptr);

31668 **DESCRIPTION**

31669 CX The functionality described on this reference page is aligned with the ISO C standard. Any
31670 conflict between the requirements described here and the ISO C standard is unintentional. This
31671 volume of POSIX.1-2017 defers to the ISO C standard.

31672 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made
31673 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the
31674 argument does not match a pointer earlier returned by a function in POSIX.1-2017 that allocates
31675 memory as if by *malloc()*, or if the space has been deallocated by a call to *free()* or *realloc()*, the
31676 behavior is undefined.

31677 Any use of a pointer that refers to freed space results in undefined behavior.

31678 **RETURN VALUE**31679 The *free()* function shall not return a value.31680 **ERRORS**

31681 No errors are defined.

31682 **EXAMPLES**

31683 None.

31684 **APPLICATION USAGE**

31685 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

31686 **RATIONALE**

31687 None.

31688 **FUTURE DIRECTIONS**

31689 None.

31690 **SEE ALSO**31691 *calloc()*, *malloc()*, *posix_memalign()*, *realloc()*

31692 XBD <stdlib.h>

31693 **CHANGE HISTORY**

31694 First released in Issue 1. Derived from Issue 1 of the SVID.

31695 **Issue 6**31696 Reference to the *valloc()* function is removed.31697 **Issue 7**

31698 The DESCRIPTION is updated to clarify that if the pointer returned is not by a function that
31699 allocates memory as if by *malloc()*, then the behavior is undefined.

31700 **NAME**

31701 freeaddrinfo, getaddrinfo — get address information

31702 **SYNOPSIS**

```

31703 #include <sys/socket.h>
31704 #include <netdb.h>
31705 void freeaddrinfo(struct addrinfo *ai);
31706 int getaddrinfo(const char *restrict nodename,
31707                const char *restrict servname,
31708                const struct addrinfo *restrict hints,
31709                struct addrinfo **restrict res);

```

31710 **DESCRIPTION**

31711 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*,
 31712 along with any additional storage associated with those structures. If the *ai_next* field of the
 31713 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall
 31714 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

31715 The *getaddrinfo()* function shall translate the name of a service location (for example, a host
 31716 name) and/or a service name and shall return a set of socket addresses and associated
 31717 information to be used in creating a socket with which to address the specified service.

31718 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,
 31719 RFC 1035, and RFC 1886.

31720 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

31721 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated
 31722 strings. One or both of these two arguments shall be supplied by the application as a non-null
 31723 pointer.

31724 The format of a valid name depends on the address family or families. If a specific family is not
 31725 given and the name could be interpreted as valid within multiple supported families, the
 31726 implementation shall attempt to resolve the name in all supported families and, in absence of
 31727 errors, one or more results shall be returned.

31728 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If
 31729 IP6 the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names
 31730 include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings
 31731 using Internet standard dot notation as specified in *inet_addr()* are valid.

31732 IP6 If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described
 31733 in *inet_ntop()* are valid.

31734 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the
 31735 requested service location is local to the caller.

31736 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If
 31737 *servname* is not null, it is a null-terminated character string identifying the requested service.
 31738 This can be either a descriptive name or a numeric representation suitable for use with the
 31739 IP6 address family or families. If the specified address family is AF_INET, AF_INET6, or
 31740 AF_UNSPEC, the service can be specified as a string specifying a decimal port number.

31741 If the *hints* argument is not null, it refers to a structure containing input values that directs the
 31742 operation by providing options and by limiting the returned information to a specific socket
 31743 type, address family, and/or protocol, as described below. The application shall ensure that each
 31744 of the *ai_addrlen*, *ai_addr*, *ai_canonname*, and *ai_next* members, as well as each of the non-standard
 31745 additional members, if any, of this *hints* structure is initialized. If any of these members has a

31746 value other than the value that would result from default initialization, the behavior is
 31747 implementation-defined. A value of AF_UNSPEC for *ai_family* means that the caller shall accept
 31748 any address family. A value of zero for *ai_socktype* means that the caller shall accept any socket
 31749 type. A value of zero for *ai_protocol* means that the caller shall accept any protocol. If *hints* is a
 31750 null pointer, the behavior shall be as if it referred to a structure containing the value zero for the
 31751 *ai_flags*, *ai_socktype*, and *ai_protocol* fields, and AF_UNSPEC for the *ai_family* field.

31752 The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-
 31753 inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME,
 31754 AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.

31755 If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in
 31756 binding a socket for accepting incoming connections for the specified service. In this case, if the
 31757 *nodename* argument is null, then the IP address portion of the socket address structure shall be
 31758 set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the
 31759 AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to
 31760 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a
 31761 connectionless protocol). In this case, if the *nodename* argument is null, then the IP address
 31762 portion of the socket address structure shall be set to the loopback address. The AI_PASSIVE
 31763 flag shall be ignored if the *nodename* argument is not null.

31764 If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function
 31765 shall attempt to determine the canonical name corresponding to *nodename* (for example, if
 31766 *nodename* is an alias or shorthand notation for a complete name).

31767 **Note:** Since different implementations use different conceptual models, the terms “canonical name”
 31768 and “alias” cannot be precisely defined for the general case. However, Domain Name System
 31769 implementations are expected to interpret them as they are used in RFC 1034.

31770 A numeric host address string is not a “name”, and thus does not have a “canonical name”
 31771 form; no address to host name translation is performed. See below for handling of the case
 31772 where a canonical name cannot be obtained.

31773 If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a
 31774 numeric host address string. Otherwise, an [EAI_NONAME] error is returned. This flag shall
 31775 prevent any type of name resolution service (for example, the DNS) from being invoked.

31776 If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a
 31777 numeric port string. Otherwise, an [EAI_NONAME] error shall be returned. This flag shall
 31778 prevent any type of name resolution service (for example, NIS+) from being invoked.

31779 IP6 By default, with an *ai_family* of AF_INET6, *getaddrinfo()* shall return only IPv6 addresses. If the
 31780 AI_V4MAPPED flag is specified along with an *ai_family* of AF_INET6, then *getaddrinfo()* shall
 31781 return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses. The
 31782 AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6. If the AI_ALL flag is
 31783 used with the AI_V4MAPPED flag, then *getaddrinfo()* shall return all matching IPv6 and IPv4
 31784 addresses. The AI_ALL flag without the AI_V4MAPPED flag shall be ignored.

31785 If the AI_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4
 31786 IP6 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6
 31787 address is configured on the local system.

31788 The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as
 31789 defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the
 31790 service name could be interpreted as valid with multiple supported socket types, the
 31791 implementation shall attempt to resolve the service name for all supported socket types and, in
 31792 the absence of errors, all possible results shall be returned. A non-zero socket type value shall

31793 limit the returned information to values with the specified socket type.

31794 If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses shall be
 31795 returned for use with any address family that can be used with the specified *nodename* and/or
 31796 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.
 31797 If *ai_family* is not AF_UNSPEC and *ai_protocol* is not zero, then addresses shall be returned for
 31798 use only with the specified address family and protocol; the value of *ai_protocol* shall be
 31799 interpreted as in a call to the *socket()* function with the corresponding values of *ai_family* and
 31800 *ai_protocol*.

31801 RETURN VALUE

31802 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value
 31803 indicates failure. The possible values for the failures are listed in the ERRORS section.

31804 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list
 31805 of **addrinfo** structures, each of which shall specify a socket address and information for use in
 31806 creating a socket with which to use that socket address. The list shall include at least one
 31807 **addrinfo** structure. The *ai_next* field of each structure contains a pointer to the next structure on
 31808 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall
 31809 include values for use with a call to the *socket()* function, and a socket address for use with the
 31810 *connect()* function or, if the AI_PASSIVE flag was specified, for use with the *bind()* function. The
 31811 fields *ai_family*, *ai_socktype*, and *ai_protocol* shall be usable as the arguments to the *socket()*
 31812 function to create a socket suitable for use with the returned address. The fields *ai_addr* and
 31813 *ai_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,
 31814 according to the AI_PASSIVE flag.

31815 If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of
 31816 the first returned **addrinfo** structure shall point to a null-terminated string containing the
 31817 canonical name corresponding to the input *nodename*; if the canonical name is not available, then
 31818 *ai_canonname* shall refer to the *nodename* argument or a string with the same contents. The
 31819 contents of the *ai_flags* field of the returned structures are undefined.

31820 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an
 31821 explicit argument (for example, *sin6_flowinfo*) shall be set to zero.

31822 **Note:** This makes it easier to compare socket address structures.

31823 ERRORS

31824 The *getaddrinfo()* function shall fail and return the corresponding error value if:

31825 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

31826 [EAI_BADFLAGS]

31827 The *flags* parameter had an invalid value.

31828 [EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

31829 [EAI_FAMILY] The address family was not recognized.

31830 [EAI_MEMORY] There was a memory allocation failure when trying to allocate storage for the
 31831 return value.

31832 [EAI_NONAME] The name does not resolve for the supplied parameters.

31833 Neither *nodename* nor *servname* were supplied. At least one of these shall be
 31834 supplied.

31835 [EAI_SERVICE] The service passed was not recognized for the specified socket type.

31836 [EAI_SOCKTYPE]
 31837 The intended socket type was not recognized.
 31838 [EAI_SYSTEM] A system error occurred; the error code can be found in *errno*.

31839 EXAMPLES

31840 The following (incomplete) program demonstrates the use of *getaddrinfo()* to obtain the socket
 31841 address structure(s) for the service named in the program's command-line argument. The
 31842 program then loops through each of the address structures attempting to create and bind a
 31843 socket to the address, until it performs a successful *bind()*.

```

31844 #include <stdio.h>
31845 #include <stdlib.h>
31846 #include <unistd.h>
31847 #include <string.h>
31848 #include <sys/socket.h>
31849 #include <netdb.h>

31850 int
31851 main(int argc, char *argv[])
31852 {
31853     struct addrinfo *result, *rp;
31854     int sfd, s;

31855     if (argc != 2) {
31856         fprintf(stderr, "Usage: %s port\n", argv[0]);
31857         exit(EXIT_FAILURE);
31858     }

31859     struct addrinfo hints = {0};
31860     hints.ai_family = AF_UNSPEC;
31861     hints.ai_socktype = SOCK_DGRAM;
31862     hints.ai_flags = AI_PASSIVE;
31863     hints.ai_protocol = 0;

31864     s = getaddrinfo(NULL, argv[1], &hints, &result);
31865     if (s != 0) {
31866         fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
31867         exit(EXIT_FAILURE);
31868     }

31869     /* getaddrinfo() returns a list of address structures.
31870     Try each address until a successful bind().
31871     If socket(2) (or bind(2)) fails, close the socket
31872     and try the next address. */

31873     for (rp = result; rp != NULL; rp = rp->ai_next) {
31874         sfd = socket(rp->ai_family, rp->ai_socktype,
31875                   rp->ai_protocol);
31876         if (sfd == -1)
31877             continue;

31878         if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
31879             break; /* Success */

31880         close(sfd);
31881     }

```



```

31882         if (rp == NULL) {           /* No address succeeded */
31883             fprintf(stderr, "Could not bind\n");
31884             exit(EXIT_FAILURE);
31885         }
31886         freeaddrinfo(result);        /* No longer needed */
31887
31887         /* ... use socket bound to sfd ... */
31888     }

```

31889 APPLICATION USAGE

31890 If the caller handles only TCP and not UDP, for example, then the *ai_protocol* member of the *hints*
 31891 structure should be set to IPPROTO_TCP when *getaddrinfo()* is called.

31892 If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the *hints* structure
 31893 should be set to AF_INET when *getaddrinfo()* is called.

31894 Although it is common practice to initialize the *hints* structure using:

```

31895 struct addrinfo hints;
31896 memset(&hints, 0, sizeof hints);

```

31897 this method is not portable according to this standard, because the structure can contain pointer
 31898 or floating-point members that are not required to have an all-bits-zero representation after
 31899 default initialization. Portable methods make use of default initialization; for example:

```

31900 struct addrinfo hints = { 0 };

```

31901 or:

```

31902 static struct addrinfo hints_init;
31903 struct addrinfo hints = hints_init;

```

31904 A future version of this standard may require that a pointer object with an all-bits-zero
 31905 representation is a null pointer, and that **addrinfo** does not have any floating-point members if a
 31906 floating-point object with an all-bits-zero representation does not have the value 0.0.

31907 The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).
 31908 It should be noted that the canonical name is a result of alias processing, and not necessarily a
 31909 unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this
 31910 in the Domain Name System context.

31911 RATIONALE

31912 None.

31913 FUTURE DIRECTIONS

31914 None.

31915 SEE ALSO

31916 [connect\(\)](#), [endservent\(\)](#), [gai_strerror\(\)](#), [getnameinfo\(\)](#), [socket\(\)](#)

31917 XBD [<netdb.h>](#), [<sys/socket.h>](#)

31918 CHANGE HISTORY

31919 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

31920 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the
 31921 ISO/IEC 9899:1999 standard.

31922 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the
 31923 DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical

- 31924 name". A reference to RFC 2181 is also added to the Informative References.
- 31925 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for
31926 alignment with IPv6. These include the following:
- 31927 Adding AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG to the allowed values for the
31928 *ai_flags* field
- 31929 Adding a description of AI_ADDRCONFIG
- 31930 Adding a description of the consequences of ignoring the AI_PASSIVE flag.
- 31931 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/39 is applied, changing "corresponding
31932 value" to "corresponding error value" in the ERRORS section.
- 31933 **Issue 7**
- 31934 Austin Group Interpretation 1003.1-2001 #013 is applied.
- 31935 Austin Group Interpretation 1003.1-2001 #146 is applied, updating the DESCRIPTION.
- 31936 An example is added.
- 31937 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0130 [939], XSH/TC2-2008/0131 [979],
31938 XSH/TC2-2008/0132 [918], and XSH/TC2-2008/0133 [934] are applied.

31939 **NAME**

31940 freelocale — free resources allocated for a locale object

31941 **SYNOPSIS**

```
31942 CX       #include <locale.h>
31943       void freelocale(locale_t locobj);
```

31944 **DESCRIPTION**

31945 The *freelocale()* function shall cause the resources allocated for a locale object returned by a call
31946 to the *newlocale()* or *duplocale()* functions to be released.

31947 The behavior is undefined if the *locobj* argument is the special locale object
31948 LC_GLOBAL_LOCALE or is not a valid locale object handle.

31949 Any use of a locale object that has been freed results in undefined behavior.

31950 **RETURN VALUE**

31951 None.

31952 **ERRORS**

31953 None.

31954 **EXAMPLES**31955 **Freeing Up a Locale Object**

31956 The following example shows a code fragment to free a locale object created by *newlocale()*:

```
31957       #include <locale.h>
31958       ...
31959       /* Every locale object allocated with newlocale() should be
31960        * freed using freelocale():
31961        */
31962       locale_t loc;
31963       /* Get the locale. */
31964       loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
31965       /* ... Use the locale object ... */
31966       ...
31967       /* Free the locale object resources. */
31968       freelocale (loc);
```

31969 **APPLICATION USAGE**

31970 None.

31971 **RATIONALE**

31972 None.

31973 **FUTURE DIRECTIONS**

31974 None.

31975 **SEE ALSO**31976 *duplocale()*, *newlocale()*, *uselocale()*31977 XBD [<locale.h>](#)

31978 **CHANGE HISTORY**

31979 First released in Issue 7.

31980 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0180 [283] is applied.

31981 **NAME**

31982 freopen — open a stream

31983 **SYNOPSIS**

31984 #include <stdio.h>

31985 FILE *freopen(const char *restrict *pathname*, const char *restrict *mode*,
31986 FILE *restrict *stream*);31987 **DESCRIPTION**31988 CX The functionality described on this reference page is aligned with the ISO C standard. Any
31989 conflict between the requirements described here and the ISO C standard is unintentional. This
31990 volume of POSIX.1-2017 defers to the ISO C standard.31991 The *freopen()* function shall first attempt to flush the stream associated with *stream* as if by a call
31992 to *fflush(stream)*. Failure to flush the stream successfully shall be ignored. If *pathname* is not a
31993 null pointer, *freopen()* shall close any file descriptor associated with *stream*. Failure to close the
31994 file descriptor successfully shall be ignored. The error and end-of-file indicators for the stream
31995 shall be cleared.31996 The *freopen()* function shall open the file whose *pathname* is the string pointed to by *pathname*
31997 and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in
31998 *fopen()*.

31999 The original stream shall be closed regardless of whether the subsequent open succeeds.

32000 If *pathname* is a null pointer, the *freopen()* function shall attempt to change the mode of the
32001 stream to that specified by *mode*, as if the name of the file currently associated with the stream
32002 had been used. In this case, the file descriptor associated with the stream need not be closed if
32003 the call to *freopen()* succeeds. It is implementation-defined which changes of mode are permitted
32004 (if any), and under what circumstances.32005 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the
32006 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an
32007 initial conversion state.32008 CX If *pathname* is not a null pointer, or if *pathname* is a null pointer and the specified mode change
32009 necessitates the file descriptor associated with the stream to be closed and reopened, the file
32010 descriptor associated with the reopened stream shall be allocated and opened as if by a call to
32011 *open()* with the following flags:

<i>freopen()</i> Mode	<i>open()</i> Flags
<i>r</i> or <i>rb</i>	O_RDONLY
<i>w</i> or <i>wb</i>	O_WRONLY O_CREAT O_TRUNC
<i>a</i> or <i>ab</i>	O_WRONLY O_CREAT O_APPEND
<i>r+</i> or <i>rb+</i> or <i>r+b</i>	O_RDWR
<i>w+</i> or <i>wb+</i> or <i>w+b</i>	O_RDWR O_CREAT O_TRUNC
<i>a+</i> or <i>ab+</i> or <i>a+b</i>	O_RDWR O_CREAT O_APPEND

32019 **RETURN VALUE**32020 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer
32021 CX shall be returned, and *errno* shall be set to indicate the error.

32022 **ERRORS**32023 The *freopen()* function shall fail if:

32024 CX [EACCES] Search permission is denied on a component of the path prefix, or the file
 32025 exists and the permissions specified by *mode* are denied, or the file does not
 32026 exist and write permission is denied for the parent directory of the file to be
 32027 created.

32028 CX [EBADF] The file descriptor underlying the stream is not a valid file descriptor when
 32029 *pathname* is a null pointer.

32030 CX [EINTR] A signal was caught during *freopen()*.

32031 CX [EISDIR] The named file is a directory and *mode* requires write access.

32032 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 32033 argument.

32034 CX [EMFILE] All file descriptors available to the process are currently open.

32035 CX [ENAMETOOLONG]

32036 The length of a component of a pathname is longer than {NAME_MAX}.

32037 CX [ENFILE] The maximum allowable number of files is currently open in the system.

32038 CX [ENOENT] The *mode* string begins with 'r' and a component of *pathname* does not name
 32039 an existing file, or *mode* begins with 'w' or 'a' and a component of the path
 32040 prefix of *pathname* does not name an existing file, or *pathname* is an empty
 32041 string.

32042 CX [ENOENT] or [ENOTDIR]

32043 The *pathname* argument contains at least one non-*<slash>* character and ends
 32044 with one or more trailing *<slash>* characters. If *pathname* without the trailing
 32045 *<slash>* characters would name an existing file, an [ENOENT] error shall not
 32046 occur.

32047 CX [ENOSPC] The directory or file system that would contain the new file cannot be
 32048 expanded, the file does not exist, and it was to be created.

32049 CX [ENOTDIR] A component of the path prefix names an existing file that is neither a
 32050 directory nor a symbolic link to a directory, or the *pathname* argument contains
 32051 at least one non-*<slash>* character and ends with one or more trailing *<slash>*
 32052 characters and the last pathname component names an existing file that is
 32053 neither a directory nor a symbolic link to a directory.

32054 CX [ENXIO] The named file is a character special or block special file, and the device
 32055 associated with this special file does not exist.

32056 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented
 32057 correctly in an object of type *off_t*.

32058 CX [EROFS] The named file resides on a read-only file system and *mode* requires write
 32059 access.

32060 The *freopen()* function may fail if:

32061 CX [EBADF] The mode with which the file descriptor underlying the stream was opened
 32062 does not support the requested mode when *pathname* is a null pointer.

32063	CX	[EINVAL]	The value of the <i>mode</i> argument is not valid.
32064	CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
32065			
32066	CX	[ENAMETOOLONG]	
32067			The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
32068			
32069			
32070	CX	[ENOMEM]	Insufficient storage space is available.
32071	CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
32072			
32073	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.
32074			

32075 EXAMPLES

32076 Directing Standard Output to a File

32077 The following example logs all standard output to the `/tmp/logfile` file.

```
32078 #include <stdio.h>
32079 ...
32080 FILE *fp;
32081 ...
32082 fp = freopen ("/tmp/logfile", "a+", stdout);
32083 ...
```

32084 APPLICATION USAGE

32085 The `freopen()` function is typically used to attach the pre-opened *streams* associated with *stdin*, *stdout*, and *stderr* to other files.

32087 Since implementations are not required to support any stream mode changes when the *pathname* argument is `NULL`, portable applications cannot rely on the use of `freopen()` to change the stream mode, and use of this feature is discouraged. The feature was originally added to the ISO C standard in order to facilitate changing *stdin* and *stdout* to binary mode. Since a 'b' character in the mode has no effect on POSIX systems, this use of the feature is unnecessary in POSIX applications. However, even though the 'b' is ignored, a successful call to `freopen(NULL, "wb", stdout)` does have an effect. In particular, for regular files it truncates the file and sets the file-position indicator for the stream to the start of the file. It is possible that these side-effects are an unintended consequence of the way the feature is specified in the ISO/IEC 9899:1999 standard, but unless or until the ISO C standard is changed, applications which successfully call `freopen(NULL, "wb", stdout)` will behave in unexpected ways on conforming systems in situations such as:

```
32099 { appl file1; appl file2; } > file3
```

32100 which will result in `file3` containing only the output from the second invocation of `appl`.

32101 RATIONALE

32102 None.

32103 **FUTURE DIRECTIONS**

32104 None.

32105 **SEE ALSO**32106 Section 2.5 (on page 495), *fclose()*, *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *mbsinit()*, *open()*,
32107 *open_memstream()*

32108 XBD <stdio.h>

32109 **CHANGE HISTORY**

32110 First released in Issue 1. Derived from Issue 1 of the SVID.

32111 **Issue 5**32112 The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the
32113 conversion state of the stream is set to an initial conversion state by a successful call to the
32114 *freopen()* function.

32115 Large File Summit extensions are added.

32116 **Issue 6**

32117 Extensions beyond the ISO C standard are marked.

32118 The following new requirements on POSIX implementations derive from alignment with the
32119 Single UNIX Specification:32120 In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open
32121 file description. This change is to support large files.32122 In the ERRORS section, the [Eoverflow] condition is added. This change is to support
32123 large files.

32124 The [ELOOP] mandatory error condition is added.

32125 A second [ENAMETOOLONG] is added as an optional error condition.

32126 The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

32127 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

32128 The *freopen()* prototype is updated.

32129 The DESCRIPTION is updated.

32130 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
32131 [ELOOP] error condition is added.

32132 The DESCRIPTION is updated regarding failure to close, changing the ``file'' to ``file descriptor''.

32133 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/40 is applied, adding the following
32134 sentence to the DESCRIPTION: ``In this case, the file descriptor associated with the stream need
32135 not be closed if the call to *freopen()* succeeds.''32136 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/41 is applied, adding an mandatory
32137 [EBADF] error, and an optional [EBADF] error to the ERRORS section.32138 **Issue 7**32139 Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the *freopen()* function
32140 allocates a file descriptor as per *open()*.

32141 Austin Group Interpretation 1003.1-2001 #143 is applied.

32142 Austin Group Interpretation 1003.1-2001 #159 is applied, clarifying requirements for the flags set
32143 on the open file description.

- 32144 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 32145 SD5-XSH-ERN-150 and SD5-XSH-ERN-219 are applied.
- 32146 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0181 [291,433], XSH/TC1-2008/0182
32147 [146,433], XSH/TC1-2008/0183 [324], and XSH/TC1-2008/0184 [14] are applied.
- 32148 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0134 [822] is applied.

32149 **NAME**

32150 frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

32151 **SYNOPSIS**

```
32152 #include <math.h>
32153 double frexp(double num, int *exp);
32154 float frexpf(float num, int *exp);
32155 long double frexpl(long double num, int *exp);
```

32156 **DESCRIPTION**

32157 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 32158 conflict between the requirements described here and the ISO C standard is unintentional. This
 32159 volume of POSIX.1-2017 defers to the ISO C standard.

32160 These functions shall break a floating-point number *num* into a normalized fraction and an
 32161 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

32162 **RETURN VALUE**

32163 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the
 32164 interval $[\frac{1}{2}, 1)$ or 0, and *num* equals *x* times 2 raised to the power **exp*.

32165 MX If *num* is NaN, a NaN shall be returned, and the value of **exp* is unspecified.

32166 If *num* is ± 0 , ± 0 shall be returned, and the value of **exp* shall be 0.

32167 If *num* is $\pm Inf$, *num* shall be returned, and the value of **exp* is unspecified.

32168 **ERRORS**

32169 No errors are defined.

32170 **EXAMPLES**

32171 None.

32172 **APPLICATION USAGE**

32173 None.

32174 **RATIONALE**

32175 None.

32176 **FUTURE DIRECTIONS**

32177 None.

32178 **SEE ALSO**

32179 [isnan\(\)](#), [ldexp\(\)](#), [modf\(\)](#)

32180 XBD [<math.h>](#)

32181 **CHANGE HISTORY**

32182 First released in Issue 1. Derived from Issue 1 of the SVID.

32183 **Issue 5**

32184 The DESCRIPTION is updated to indicate how an application should check for an error. This
 32185 text was previously published in the APPLICATION USAGE section.

32186 **Issue 6**

32187 The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999
32188 standard.

32189 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
32190 revised to align with the ISO/IEC 9899:1999 standard.

32191 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
32192 marked.

32193 **NAME**

32194 fscanf, scanf, sscanf ‡convert formatted input

32195 **SYNOPSIS**

32196 #include <stdio.h>

32197 int fscanf(FILE *restrict stream, const char *restrict format, ...);

32198 int scanf(const char *restrict format, ...);

32199 int sscanf(const char *restrict s, const char *restrict format, ...);

32200 **DESCRIPTION**

32201 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 32202 conflict between the requirements described here and the ISO C standard is unintentional. This
 32203 volume of POSIX.1-2017 defers to the ISO C standard.

32204 The *fscanf()* function shall read from the named input *stream*. The *scanf()* function shall read
 32205 from the standard input stream *stdin*. The *sscanf()* function shall read from the string *s*. Each
 32206 function reads bytes, interprets them according to a *format*, and stores the results in its
 32207 arguments. Each expects, as arguments, a control string *format* described below, and a set of
 32208 *pointer* arguments indicating where the converted input should be stored. The result is
 32209 undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while
 32210 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

32211 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 32212 to the next unused argument. In this case, the conversion specifier character % (see below) is
 32213 replaced by the sequence "%n\$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}].
 32214 This feature provides for the definition of format strings that select arguments in an order
 32215 appropriate to specific languages. In format strings containing the "%n\$" form of conversion
 32216 specifications, it is unspecified whether numbered arguments in the argument list can be
 32217 referenced from the format string more than once.

32218 The *format* can contain either form of a conversion specification ‡that is, % or "%n\$" ‡but the
 32219 two forms cannot be mixed within a single *format* string. The only exception to this is that %% or
 32220 %* can be mixed with the "%n\$" form. When numbered argument specifications are used,
 32221 specifying the *N*th argument requires that all the leading arguments, from the first to the
 32222 (*N*-1)th, are pointers.

32223 The *fscanf()* function in all its forms shall allow detection of a language-dependent radix
 32224 character in the input string. The radix character is defined in the current locale (category
 32225 LC_NUMERIC). In the POSIX locale, or in a locale where the radix character is not defined, the
 32226 radix character shall default to a <period> ('.').

32227 The *format* is a character string, beginning and ending in its initial shift state, if any, composed
 32228 of zero or more directives. Each directive is composed of one of the following: one or more
 32229 white-space characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary
 32230 character (neither '%' nor a white-space character); or a conversion specification. Each
 32231 CX conversion specification is introduced by the character '%' or the character sequence "%n\$",
 32232 after which the following appear in sequence:

32233 An optional assignment-suppressing character '*'.

32234 An optional non-zero decimal integer that specifies the maximum field width.

32235 CX An optional assignment-allocation character 'm'.

32236 An option length modifier that specifies the size of the receiving object.

32237 A *conversion specifier* character that specifies the type of conversion to be applied. The valid
32238 conversion specifiers are described below.

32239 The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as
32240 detailed below, the function shall return. Failures are described as input failures (due to the
32241 unavailability of input bytes) or matching failures (due to inappropriate input).

32242 A directive composed of one or more white-space characters shall be executed by reading input
32243 until no more valid input can be read, or up to the first byte which is not a white-space character,
32244 which remains unread.

32245 A directive that is an ordinary character shall be executed as follows: the next byte shall be read
32246 from the input and compared with the byte that comprises the directive; if the comparison
32247 shows that they are not equivalent, the directive shall fail, and the differing and subsequent
32248 bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a
32249 character from being read, the directive shall fail.

32250 A directive that is a conversion specification defines a set of matching input sequences, as
32251 described below for each conversion character. A conversion specification shall be executed in
32252 the following steps.

32253 Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion
32254 specification includes a `[`, `c`, `C`, or `n` conversion specifier.

32255 An item shall be read from the input, unless the conversion specification includes an `n`
32256 conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to
32257 any specified maximum field width, which may be measured in characters or bytes dependent
32258 on the conversion specifier) which is an initial subsequence of a matching sequence. The first
32259 byte, if any, after the input item shall remain unread. If the length of the input item is 0, the
32260 execution of the conversion specification shall fail; this condition is a matching failure, unless
32261 end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is
32262 an input failure.

32263 Except in the case of a `%` conversion specifier, the input item (or, in the case of a `%n` conversion
32264 specification, the count of input bytes) shall be converted to a type appropriate to the conversion
32265 character. If the input item is not a matching sequence, the execution of the conversion
32266 specification fails; this condition is a matching failure. Unless assignment suppression was
32267 indicated by a `'*'`, the result of the conversion shall be placed in the object pointed to by the
32268 first argument following the *format* argument that has not already received a conversion result if
32269 CX the conversion specification is introduced by `%`, or in the *n*th argument if introduced by the
32270 character sequence `"%n$"`. If this object does not have an appropriate type, or if the result of the
32271 conversion cannot be represented in the space provided, the behavior is undefined.

32272 CX The `%c`, `%s`, and `%[` conversion specifiers shall accept an optional assignment-allocation
32273 character `'m'`, which shall cause a memory buffer to be allocated to hold the string converted
32274 including a terminating null character. In such a case, the argument corresponding to the
32275 conversion specifier should be a reference to a pointer variable that will receive a pointer to the
32276 allocated buffer. The system shall allocate a buffer as if *malloc()* had been called. The application
32277 shall be responsible for freeing the memory after usage. If there is insufficient memory to
32278 allocate a buffer, the function shall set *errno* to `[ENOMEM]` and a conversion error shall result. If
32279 the function returns EOF, any memory successfully allocated for parameters using assignment-
32280 allocation character `'m'` by this call shall be freed before the function returns.

32281		The length modifiers and their meanings are:
32282	hh	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to signed char or unsigned char .
32283		
32284	h	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to short or unsigned short .
32285		
32286	l (ell)	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to long or unsigned long ; that a following <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , or <code>G</code> conversion specifier applies to an argument with type pointer to double ; or that a following <code>c</code> , <code>s</code> , or <code>[]</code> conversion specifier applies to an argument with type pointer to wchar_t . If the <code>'m'</code> assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to wchar_t .
32287		
32288		
32289		
32290	CX	
32291		
32292	ll (ell-ell)	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to long long or unsigned long long .
32293		
32294		
32295	j	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to intmax_t or uintmax_t .
32296		
32297	z	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to size_t or the corresponding signed integer type.
32298		
32299	t	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , or <code>n</code> conversion specifier applies to an argument with type pointer to ptrdiff_t or the corresponding unsigned type.
32300		
32301	L	Specifies that a following <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , or <code>G</code> conversion specifier applies to an argument with type pointer to long double .
32302		
32303		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
32304		
32305		The following conversion specifiers are valid:
32306	d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <code>strtol()</code> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
32307		
32308		
32309		
32310	i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <code>strtol()</code> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
32311		
32312		
32313		
32314	o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
32315		
32316		
32317		
32318	u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
32319		
32320		
32321		
32322	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding
32323		
32324		

32325 argument is a pointer to **unsigned**.

32326 a, e, f, g

32327 Matches an optionally signed floating-point number, infinity, or NaN, whose format is

32328 the same as expected for the subject sequence of *strtod()*. In the absence of a size

32329 modifier, the application shall ensure that the corresponding argument is a pointer to

32330 **float**.

32331 If the *fprintf()* family of functions generates character string representations for infinity

32332 and NaN (a symbolic entity encoded in floating-point format) to support

32333 IEEE Std 754-1985, the *fscanf()* family of functions shall recognize them as input.

32334 s

32335 Matches a sequence of bytes that are not white-space characters. If the 'm' assignment-

32336 allocation character is not specified, the application shall ensure that the corresponding

32337 argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned**

32338 **char** large enough to accept the sequence and a terminating null character code, which

32339 **CX** shall be added automatically. Otherwise, the application shall ensure that the

32340 corresponding argument is a pointer to a pointer to a **char**.

32341 If an l (ell) qualifier is present, the input is a sequence of characters that begins in the

32342 initial shift state. Each character shall be converted to a wide character as if by a call to

32343 the *mbrtowc()* function, with the conversion state described by an **mbstate_t** object

32344 initialized to zero before the first character is converted. If the 'm' assignment-

32345 allocation character is not specified, the application shall ensure that the corresponding

32346 argument is a pointer to an array of **wchar_t** large enough to accept the sequence and

32347 **CX** the terminating null wide character, which shall be added automatically. Otherwise,

32348 the application shall ensure that the corresponding argument is a pointer to a pointer to

32349 a **wchar_t**.

32350 [

32351 Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The

32352 normal skip over white-space characters shall be suppressed in this case. If the 'm'

32353 assignment-allocation character is not specified, the application shall ensure that the

32354 corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**,

32355 **CX** or **unsigned char** large enough to accept the sequence and a terminating null byte,

32356 which shall be added automatically. Otherwise, the application shall ensure that the

32357 corresponding argument is a pointer to a pointer to a **char**.

32358 If an l (ell) qualifier is present, the input is a sequence of characters that begins in the

32359 initial shift state. Each character in the sequence shall be converted to a wide character

32360 as if by a call to the *mbrtowc()* function, with the conversion state described by an

32361 **mbstate_t** object initialized to zero before the first character is converted. If the 'm'

32362 assignment-allocation character is not specified, the application shall ensure that the

32363 corresponding argument is a pointer to an array of **wchar_t** large enough to accept the

32364 **CX** sequence and the terminating null wide character, which shall be added automatically.

32365 Otherwise, the application shall ensure that the corresponding argument is a pointer to

32366 a pointer to a **wchar_t**.

32367 The conversion specification includes all subsequent bytes in the *format* string up to

32368 and including the matching <right-square-bracket> (']'). The bytes between the

32369 square brackets (the *scanlist*) comprise the *scanset*, unless the byte after the <left-

32370 square-bracket> is a <circumflex> ('^'), in which case the *scanset* contains all bytes

32371 that do not appear in the *scanlist* between the <circumflex> and the <right-square-

32372 bracket>. If the conversion specification begins with "[]" or "[^]", the <right-

32373		first <right-square-bracket> is the one that ends the conversion specification. If a '-' is
32374		in the scanlist and is not the first character, nor the second where the first character is a
32375		'^', nor the last character, the behavior is implementation-defined.
32376	c	Matches a sequence of bytes of the number specified by the field width (1 if no field
32377		width is present in the conversion specification). No null byte is added. The normal
32378		skip over white-space characters shall be suppressed in this case. If the 'm'
32379		assignment-allocation character is not specified, the application shall ensure that the
32380		corresponding argument is a pointer to the initial byte of an array of char , signed char ,
32381	CX	or unsigned char large enough to accept the sequence. Otherwise, the application
32382		shall ensure that the corresponding argument is a pointer to a pointer to a char .
32383		If an l (ell) qualifier is present, the input shall be a sequence of characters that begins in
32384		the initial shift state. Each character in the sequence is converted to a wide character as
32385		if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an
32386		mbstate_t object initialized to zero before the first character is converted. No null wide
32387		character is added. If the 'm' assignment-allocation character is not specified, the
32388		application shall ensure that the corresponding argument is a pointer to an array of
32389	CX	wchar_t large enough to accept the resulting sequence of wide characters. Otherwise,
32390		the application shall ensure that the corresponding argument is a pointer to a pointer to
32391		a wchar_t .
32392	p	Matches an implementation-defined set of sequences, which shall be the same as the set
32393		of sequences that is produced by the %p conversion specification of the corresponding
32394		<i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a
32395		pointer to a pointer to void . The interpretation of the input item is implementation-
32396		defined. If the input item is a value converted earlier during the same program
32397		execution, the pointer that results shall compare equal to that value; otherwise, the
32398		behavior of the %p conversion specification is undefined.
32399	n	No input is consumed. The application shall ensure that the corresponding argument is
32400		a pointer to the integer into which shall be written the number of bytes read from the
32401		input so far by this call to the <i>fscanf()</i> functions. Execution of a %n conversion
32402		specification shall not increment the assignment count returned at the completion of
32403		execution of the function. No argument shall be converted, but one shall be consumed.
32404		If the conversion specification includes an assignment-suppressing character or a field
32405		width, the behavior is undefined.
32406	XSI	C Equivalent to lc .
32407	XSI	S Equivalent to ls .
32408	%	Matches a single '%' character; no conversion or assignment occurs. The complete
32409		conversion specification shall be %%. If a conversion specification is invalid, the behavior is undefined.
32410		
32411		The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to a, e, f, g, and
32412		x, respectively.
32413		If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs
32414		before any bytes matching the current conversion specification (except for %n) have been read
32415		(other than leading white-space characters, where permitted), execution of the current
32416		conversion specification shall terminate with an input failure. Otherwise, unless execution of the
32417		current conversion specification is terminated with a matching failure, execution of the
32418		following conversion specification (if any) shall be terminated with an input failure.

32419 Reaching the end of the string in *sscanf()* shall be equivalent to encountering end-of-file for
32420 *fscanf()*.

32421 If conversion terminates on a conflicting input, the offending input is left unread in the input.
32422 Any trailing white space (including <newline> characters) shall be left unread unless matched
32423 by a conversion specification. The success of literal matches and suppressed assignments is only
32424 directly determinable via the %n conversion specification.

32425 CX The *fscanf()* and *scanf()* functions may mark the last data access timestamp of the file associated
32426 with *stream* for update. The last data access timestamp shall be marked for update by the first
32427 successful execution of *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*,
32428 *fscanf()*, or *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

32429 RETURN VALUE

32430 Upon successful completion, these functions shall return the number of successfully matched
32431 and assigned input items; this number can be zero in the event of an early matching failure. If
32432 the input ends before the first conversion (if any) has completed, and without a matching failure
32433 having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has
32434 CX completed, and without a matching failure having occurred, EOF shall be returned and *errno*
32435 shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be
32436 set.

32437 ERRORS

32438 For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or
32439 *fgetwc()*.

32440 In addition, the *fscanf()* function shall fail if:

32441 CX [EILSEQ] Input byte sequence does not form a valid character.

32442 [ENOMEM] Insufficient storage space is available.

32443 In addition, the *fscanf()* function may fail if:

32444 CX [EINVAL] There are insufficient arguments.

32445 EXAMPLES

32446 The call:

```
32447 int i, n; float x; char name[50];
32448 n = scanf("%d%f%s", &i, &x, name);
```

32449 with the input line:

```
32450 25 54.32E-1 Hamster
```

32451 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
32452 "Hamster".

32453 The call:

```
32454 int i; float x; char name[50];
32455 (void) scanf("%2d%f%d %[0123456789]", &i, &x, name);
```

32456 with input:

```
32457 56789 0123 56a72
```

32458 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to
32459 *getchar()* shall return the character 'a'.

32460 **Reading Data into an Array**

32461 The following call uses *fscanf()* to read three floating-point numbers from standard input into
 32462 the *input* array.

```
32463 float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

32464 **APPLICATION USAGE**

32465 If the application calling *fscanf()* has any objects of type **wint_t** or **wchar_t**, it must also include
 32466 the **<wchar.h>** header to have these objects defined.

32467 For functions that allocate memory as if by *malloc()*, the application should release such memory
 32468 when it is no longer required by a call to *free()*. For *fscanf()*, this is memory allocated via use of
 32469 the 'm' assignment-allocation character.

32470 **RATIONALE**

32471 This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few “obvious”
 32472 things were not included. Specifically, the set of characters allowed in a scanset is limited to
 32473 single-byte characters. In other similar places, multi-byte characters have been permitted, but
 32474 for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications
 32475 needing this could use the corresponding wide-character functions to achieve the desired
 32476 results.

32477 **FUTURE DIRECTIONS**

32478 None.

32479 **SEE ALSO**

32480 [Section 2.5](#) (on page 495), *fprintf()*, *getc()*, *setlocale()*, *strtod()*, *strtol()*, *strtoul()*, *wcrtomb()*

32481 [XBD Chapter 7](#) (on page 135), **<inttypes.h>**, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

32482 **CHANGE HISTORY**

32483 First released in Issue 1. Derived from Issue 1 of the SVID.

32484 **Issue 5**

32485 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the *l* (ell) qualifier is
 32486 now defined for the *c*, *s*, and *[* conversion specifiers.

32487 The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the
 32488 *fprintf()* family of functions, then they are recognized by the *fscanf()* family.

32489 **Issue 6**

32490 The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several
 32491 occurrences of “characters” in the text which have been replaced with the term “bytes”.

32492 The normative text is updated to avoid use of the term “must” for application requirements.

32493 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

32494 The prototypes for *fscanf()*, *scanf()*, and *sscanf()* are updated.

32495 The DESCRIPTION is updated.

32496 The *hh*, *ll*, *j*, *t*, and *z* length modifiers are added.

32497 The *a*, *A*, and *F* conversion characters are added.

32498 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
 32499 specification” consistently.

32500 **Issue 7**

- 32501 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 32502 SD5-XSH-ERN-9 is applied, correcting *fscanf()* to *scanf()* in the DESCRIPTION.
- 32503 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character 'm'.
- 32504 Functionality relating to the %n\$ form of conversion specification is moved from the XSI option
32505 to the Base.
- 32506 Changes are made related to support for finegrained timestamps.
- 32507 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
32508 *malloc()*.
- 32509 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0185 [302], XSH/TC1-2008/0186 [90],
32510 and XSH/TC1-2008/0187 [14] are applied. XSH/TC1-2008/0186 [90] changes the second
32511 sentence in the RETURN VALUE section to align with expected wording changes in the next
32512 revision of the ISO C standard.
- 32513 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0135 [936] is applied.

32514 **NAME**

32515 fseek, fseeko — reposition a file-position indicator in a stream

32516 **SYNOPSIS**

32517 #include <stdio.h>

32518 int fseek(FILE *stream, long offset, int whence);

32519 CX int fseeko(FILE *stream, off_t offset, int whence);

32520 **DESCRIPTION**32521 CX The functionality described on this reference page is aligned with the ISO C standard. Any
32522 conflict between the requirements described here and the ISO C standard is unintentional. This
32523 volume of POSIX.1-2017 defers to the ISO C standard.32524 The *fseek()* function shall set the file-position indicator for the stream pointed to by *stream*. If a
32525 read or write error occurs, the error indicator for the stream shall be set and *fseek()* fails.32526 The new position, measured in bytes from the beginning of the file, shall be obtained by adding
32527 *offset* to the position specified by *whence*. The specified point is the beginning of the file for
32528 SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for
32529 SEEK_END.32530 If the stream is to be used with wide-character input/output functions, the application shall
32531 ensure that *offset* is either 0 or a value returned by an earlier call to *ftell()* on the same stream and
32532 *whence* is SEEK_SET.32533 A successful call to *fseek()* shall clear the end-of-file indicator for the stream and undo any effects
32534 of *ungetc()* and *ungetwc()* on the same stream. After an *fseek()* call, the next operation on an
32535 update stream may be either input or output.32536 CX If the most recent operation, other than *ftell()*, on a given stream is *fflush()*, the file offset in the
32537 underlying open file description shall be adjusted to reflect the location specified by *fseek()*.32538 The *fseek()* function shall allow the file-position indicator to be set beyond the end of existing
32539 data in the file. If data is later written at this point, subsequent reads of data in the gap shall
32540 return bytes with the value 0 until data is actually written into the gap.32541 The behavior of *fseek()* on devices which are incapable of seeking is implementation-defined.
32542 The value of the file offset associated with such a device is undefined.32543 If the stream is writable and buffered data had not been written to the underlying file, *fseek()*
32544 shall cause the unwritten data to be written to the file and shall mark the last data modification
32545 and last file status change timestamps of the file for update.32546 In a locale with state-dependent encoding, whether *fseek()* restores the stream's shift state is
32547 implementation-defined.32548 The *fseeko()* function shall be equivalent to the *fseek()* function except that the *offset* argument is
32549 of type **off_t**.32550 **RETURN VALUE**32551 CX The *fseek()* and *fseeko()* functions shall return 0 if they succeed.32552 CX Otherwise, they shall return -1 and set *errno* to indicate the error.32553 **ERRORS**32554 CX The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream's*
32555 buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or
32556 *write()* to be invoked, and:

32557	CX	[EAGAIN]	The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the write operation.
32558			
32559	CX	[EBADF]	The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.
32560			
32561	CX	[EFBIG]	An attempt was made to write a file that exceeds the maximum file size.
32562	XSI	[EFBIG]	An attempt was made to write a file that exceeds the file size limit of the process.
32563			
32564	CX	[EFBIG]	The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.
32565			
32566	CX	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
32567			
32568	CX	[EINVAL]	The <i>whence</i> argument is invalid. The resulting file-position indicator would be set to a negative value.
32569			
32570	CX	[EIO]	A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a <i>write()</i> to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
32571			
32572			
32573			
32574			
32575	CX	[ENOSPC]	There was no free space remaining on the device containing the file.
32576	CX	[EOVERFLOW]	For <i>fseek()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type long .
32577			
32578	CX	[EOVERFLOW]	For <i>fseeko()</i> , the resulting file offset would be a value which cannot be represented correctly in an object of type off_t .
32579			
32580	CX	[EPIPE]	An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.
32581			
32582	CX	[ESPIPE]	The file descriptor underlying <i>stream</i> is associated with a pipe, FIFO, or socket.
32583	CX		The <i>fseek()</i> and <i>fseeko()</i> functions may fail if:
32584	CX	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
32585			

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 495), *fopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit()*, *ungetc()*, *write()*

XBD <stdio.h>

CHANGE HISTORY

32598 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

32601 Normative text previously in the APPLICATION USAGE section is moved to the
32602 DESCRIPTION.

32603 Large File Summit extensions are added.

Issue 6

32604 Extensions beyond the ISO C standard are marked.

32606 The following new requirements on POSIX implementations derive from alignment with the
32607 Single UNIX Specification:

32608 The *fseeko()* function is added.

32609 The [EFBIG], [Eoverflow], and [ENXIO] mandatory error conditions are added.

32610 The following change is incorporated for alignment with the FIPS requirements:

32611 The [EINTR] error is no longer an indication that the implementation does not report
32612 partial transfers.

32613 The normative text is updated to avoid use of the term “must” for application requirements.

32614 The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and
32615 then on error the error indicator is set and *fseek()* fails. This is for alignment with the
32616 ISO/IEC 9899:1999 standard.

32617 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/42 is applied, updating the [EAGAIN]
32618 error in the ERRORS section from “the process would be delayed” to “the thread would be
32619 delayed”.

Issue 7

32620 Changes are made related to support for finegrained timestamps.

32622 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0188 [79], XSH/TC1-2008/0189 [122],
32623 XSH/TC1-2008/0190 [225], and XSH/TC1-2008/0191 [14] are applied.

32624 **NAME**

32625 fsetpos — set current file position

32626 **SYNOPSIS**

32627 #include <stdio.h>

32628 int fsetpos(FILE *stream, const fpos_t *pos);

32629 **DESCRIPTION**

32630 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 32631 conflict between the requirements described here and the ISO C standard is unintentional. This
 32632 volume of POSIX.1-2017 defers to the ISO C standard.

32633 The *fsetpos()* function shall set the file position and state indicators for the stream pointed to by
 32634 *stream* according to the value of the object pointed to by *pos*, which the application shall ensure is
 32635 a value obtained from an earlier call to *fgetpos()* on the same stream. If a read or write error
 32636 occurs, the error indicator for the stream shall be set and *fsetpos()* fails.

32637 A successful call to the *fsetpos()* function shall clear the end-of-file indicator for the stream and
 32638 undo any effects of *ungetc()* on the same stream. After an *fsetpos()* call, the next operation on an
 32639 update stream may be either input or output.

32640 CX The behavior of *fsetpos()* on devices which are incapable of seeking is implementation-defined.
 32641 The value of the file offset associated with such a device is undefined.

32642 The *fsetpos()* function shall not change the setting of *errno* if successful.

32643 **RETURN VALUE**

32644 The *fsetpos()* function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and
 32645 set *errno* to indicate the error.

32646 **ERRORS**

32647 CX The *fsetpos()* function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to
 32648 be flushed, and the call to *fsetpos()* causes an underlying *lseek()* or *write()* to be invoked, and:

32649 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor and the thread would be
 32650 delayed in the write operation.

32651 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
 32652 stream's buffer needed to be flushed and the file is not open.

32653 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

32654 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 32655 process.

32656 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 32657 offset maximum associated with the corresponding stream.

32658 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
 32659 was transferred.

32660 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
 32661 process group attempting to perform a *write()* to its controlling terminal,
 32662 TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not
 32663 ignoring SIGTTOU, and the process group of the process is orphaned. This
 32664 error may also be returned under implementation-defined conditions.

32665 CX [ENOSPC] There was no free space remaining on the device containing the file.

32666 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
32667 by any process; a SIGPIPE signal shall also be sent to the thread.

32668 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

32669 The *fsetpos()* function may fail if:

32670 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
32671 capabilities of the device.

32672 EXAMPLES

32673 None.

32674 APPLICATION USAGE

32675 None.

32676 RATIONALE

32677 None.

32678 FUTURE DIRECTIONS

32679 None.

32680 SEE ALSO

32681 [Section 2.5](#) (on page 495), [fopen\(\)](#), [ftell\(\)](#), [lseek\(\)](#), [rewind\(\)](#), [ungetc\(\)](#), [write\(\)](#)

32682 XBD [<stdio.h>](#)

32683 CHANGE HISTORY

32684 First released in Issue 4. Derived from the ISO C standard.

32685 Issue 6

32686 Extensions beyond the ISO C standard are marked.

32687 The normative text is updated to avoid use of the term “must” for application requirements.

32688 The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or
32689 write error. This is for alignment with the ISO/IEC 9899:1999 standard.

32690 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous
32691 [EINVAL] error case from the ERRORS section.

32692 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/43 is applied, updating the [EAGAIN]
32693 error in the ERRORS section from “the process would be delayed” to “the thread would be
32694 delayed”.

32695 Issue 7

32696 SD5-XSH-ERN-220 is applied.

32697 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0192 [105], XSH/TC1-2008/0193 [79],
32698 XSH/TC1-2008/0194 [225], XSH/TC1-2008/0195 [450], XSH/TC1-2008/0196 [450], and
32699 XSH/TC1-2008/0197 [14] are applied.

32700 **NAME**

32701 fstat ‡get file status

32702 **SYNOPSIS**

32703 #include <sys/stat.h>

32704 int fstat(int *fildev*, struct stat **buf*);32705 **DESCRIPTION**32706 The *fstat()* function shall obtain information about an open file associated with the file
32707 descriptor *fildev*, and shall write it to the area pointed to by *buf*.32708 SHM If *fildev* references a shared memory object, the implementation shall update in the **stat** structure
32709 pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
32710 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
32711 valid. The implementation may update other fields and flags.32712 TYM If *fildev* references a typed memory object, the implementation shall update in the **stat** structure
32713 pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
32714 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
32715 valid. The implementation may update other fields and flags.32716 The *buf* argument is a pointer to a **stat** structure, as defined in <sys/stat.h>, into which
32717 information is placed concerning the file.32718 For all other file types defined in this volume of POSIX.1-2017, the structure members *st_mode*,
32719 *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the
32720 value of the *st_nlink* member shall be set to the number of links to the file.32721 An implementation that provides additional or alternative file access control mechanisms may,
32722 under implementation-defined conditions, cause *fstat()* to fail.32723 The *fstat()* function shall update any time-related fields (as described in XBD [Section 4.9](#), on
32724 page 109), before writing into the **stat** structure.32725 **RETURN VALUE**32726 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
32727 indicate the error.32728 **ERRORS**32729 The *fstat()* function shall fail if:32730 [EBADF] The *fildev* argument is not a valid file descriptor.

32731 [EIO] An I/O error occurred while reading from the file system.

32732 [E_OVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
32733 serial number cannot be represented correctly in the structure pointed to by
32734 *buf*.32735 The *fstat()* function may fail if:32736 [E_OVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*
32737 argument.

32738 **EXAMPLES**32739 **Obtaining File Status Information**

32740 The following example shows how to obtain file status information for a file named
 32741 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The
 32742 **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file
 32743 descriptor *fildev*.

```
32744 #include <sys/types.h>
32745 #include <sys/stat.h>
32746 #include <fcntl.h>

32747 struct stat buffer;
32748 int      status;
32749 ...
32750 fildev = open("/home/cnd/mod1", O_RDWR);
32751 status = fstat(fildev, &buffer);
```

32752 **APPLICATION USAGE**

32753 None.

32754 **RATIONALE**

32755 None.

32756 **FUTURE DIRECTIONS**

32757 None.

32758 **SEE ALSO**

32759 [fstatat\(\)](#)

32760 XBD [Section 4.9](#) (on page 109), [<sys/stat.h>](#), [<sys/types.h>](#)

32761 **CHANGE HISTORY**

32762 First released in Issue 1. Derived from Issue 1 of the SVID.

32763 **Issue 5**

32764 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

32765 Large File Summit extensions are added.

32766 **Issue 6**

32767 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

32768 The following new requirements on POSIX implementations derive from alignment with the
 32769 Single UNIX Specification:

32770 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 32771 required for conforming implementations of previous POSIX specifications, it was not
 32772 required for UNIX applications.

32773 The [EIO] mandatory error condition is added.

32774 The [EOVERFLOW] mandatory error condition is added. This change is to support large
 32775 files.

32776 The [EOVERFLOW] optional error condition is added.

32777 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 32778 shared memory object semantics apply to typed memory objects.

32779 **Issue 7**

32780 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink*
32781 applies.

32782 Changes are made related to support for finegrained timestamps.

32783 **NAME**

32784 fstatat, lstat, stat ‡get file status

32785 **SYNOPSIS**

32786 OH #include <fcntl.h>

32787 #include <sys/stat.h>

32788 int fstatat(int fd, const char *restrict path,

32789 struct stat *restrict buf, int flag);

32790 int lstat(const char *restrict path, struct stat *restrict buf);

32791 int stat(const char *restrict path, struct stat *restrict buf);

32792 **DESCRIPTION**

32793 The *stat()* function shall obtain information about the named file and write it to the area pointed
 32794 to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or
 32795 execute permission of the named file is not required. An implementation that provides
 32796 additional or alternate file access control mechanisms may, under implementation-defined
 32797 conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file
 32798 specified by *path*.

32799 If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using
 32800 the contents of the symbolic link, and shall return information pertaining to the resulting file if
 32801 the file exists.

32802 The *buf* argument is a pointer to a **stat** structure, as defined in the <sys/stat.h> header, into
 32803 which information is placed concerning the file.

32804 The *stat()* function shall update any time-related fields (as described in XBD Section 4.9, on page
 32805 109), before writing into the **stat** structure.

32806 SHM If the named file is a shared memory object, the implementation shall update in the **stat** structure
 32807 pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
 32808 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
 32809 valid. The implementation may update other fields and flags.

32810 TYM If the named file is a typed memory object, the implementation shall update in the **stat** structure
 32811 pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the
 32812 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be
 32813 valid. The implementation may update other fields and flags.

32814 For all other file types defined in this volume of POSIX.1-2017, the structure members *st_mode*,
 32815 *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the
 32816 value of the member *st_nlink* shall be set to the number of links to the file.

32817 The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In
 32818 that case *lstat()* shall return information about the link, while *stat()* shall return information
 32819 about the file the link references.

32820 For symbolic links, the *st_mode* member shall contain meaningful information when used with
 32821 the file type macros. The file mode bits in *st_mode* are unspecified. The structure members *st_ino*,
 32822 *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of
 32823 the *st_nlink* member shall be set to the number of (hard) links to the symbolic link. The value of
 32824 the *st_size* member shall be set to the length of the pathname contained in the symbolic link not
 32825 including any terminating null byte.

32826 The *fstatat()* function shall be equivalent to the *stat()* or *lstat()* function, depending on the value
 32827 of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status
 32828 shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead

32829 of the current working directory. If the access mode of the open file description associated with
 32830 the file descriptor is not O_SEARCH, the function shall check whether directory searches are
 32831 permitted using the current permissions of the directory underlying the file descriptor. If the
 32832 access mode is O_SEARCH, the function shall not perform the check.

32833 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 32834 in `<fcntl.h>`:

32835 AT_SYMLINK_NOFOLLOW

32836 If *path* names a symbolic link, the status of the symbolic link is returned.

32837 If *fstatat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 32838 directory shall be used and the behavior shall be identical to a call to *stat()* or *lstat()* respectively,
 32839 depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in *flag*.

32840 RETURN VALUE

32841 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 32842 return -1 and set *errno* to indicate the error.

32843 ERRORS

32844 These functions shall fail if:

32845 [EACCES] Search permission is denied for a component of the path prefix.

32846 [EIO] An error occurred while reading from the file system.

32847 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 32848 argument.

32849 [ENAMETOOLONG]

32850 The length of a component of a pathname is longer than {NAME_MAX}.

32851 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

32852 [ENOTDIR] A component of the path prefix names an existing file that is neither a
 32853 directory nor a symbolic link to a directory, or the *path* argument contains at
 32854 least one non-`<slash>` character and ends with one or more trailing `<slash>`
 32855 characters and the last pathname component names an existing file that is
 32856 neither a directory nor a symbolic link to a directory.

32857 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file
 32858 serial number cannot be represented correctly in the structure pointed to by
 32859 *buf*.

32860 The *fstatat()* function shall fail if:

32861 [EACCES] The access mode of the open file description associated with *fd* is not
 32862 O_SEARCH and the permissions of the directory underlying *fd* do not permit
 32863 directory searches.

32864 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 32865 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

32866 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
 32867 with a non-directory file.

32868 These functions may fail if:

32869 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 32870 resolution of the *path* argument.

- 32871 [ENAMETOOLONG]
 32872 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 32873 symbolic link produced an intermediate result with a length that exceeds
 32874 {PATH_MAX}.
- 32875 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.
- 32876 The *fstatat()* function may fail if:
- 32877 [EINVAL] The value of the *flag* argument is not valid.

32878 EXAMPLES

32879 Obtaining File Status Information

32880 The following example shows how to obtain file status information for a file named
 32881 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
32882 #include <sys/types.h>
32883 #include <sys/stat.h>
32884 #include <fcntl.h>

32885 struct stat buffer;
32886 int      status;
32887 ...
32888 status = stat("/home/cnd/mod1", &buffer);
```

32889 Getting Directory Information

32890 The following example fragment gets status information for each entry in a directory. The call to
 32891 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines
 32892 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the
 32893 program.

```
32894 #include <sys/types.h>
32895 #include <sys/stat.h>
32896 #include <dirent.h>
32897 #include <pwd.h>
32898 #include <grp.h>
32899 #include <time.h>
32900 #include <locale.h>
32901 #include <langinfo.h>
32902 #include <stdio.h>
32903 #include <stdint.h>

32904 struct dirent *dp;
32905 struct stat   statbuf;
32906 struct passwd *pwd;
32907 struct group  *grp;
32908 struct tm     *tm;
32909 char          datestring[256];
32910 ...
32911 /* Loop through directory entries. */
32912 while ((dp = readdir(dir)) != NULL) {

32913     /* Get entry's information. */
32914     if (stat(dp->d_name, &statbuf) == -1)
```

```

32915         continue;
32916         /* Print out type, permissions, and number of links. */
32917         printf("%10.10s", spem (statbuf.st_mode));
32918         printf("%4d", statbuf.st_nlink);
32919
32919         /* Print out owner's name if it is found using getpwuid(). */
32920         if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
32921             printf(" %-8.8s", pwd->pw_name);
32922         else
32923             printf(" %-8d", statbuf.st_uid);
32924
32924         /* Print out group name if it is found using getgrgid(). */
32925         if ((grp = getgrgid(statbuf.st_gid)) != NULL)
32926             printf(" %-8.8s", grp->gr_name);
32927         else
32928             printf(" %-8d", statbuf.st_gid);
32929
32929         /* Print size of file. */
32930         printf(" %9jd", (intmax_t)statbuf.st_size);
32931
32931         tm = localtime(&statbuf.st_mtime);
32932
32932         /* Get localized date string. */
32933         strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
32934         printf(" %s %s\n", datestring, dp->d_name);
32935     }

```

32936 Obtaining Symbolic Link Status Information

32937 The following example shows how to obtain status information for a symbolic link named
32938 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*
32939 argument specified the pathname for the file pointed to by the symbolic link (**/home/cnd/mod1**),
32940 the results of calling the function would be the same as those returned by a call to the *stat()*
32941 function.

```

32942 #include <sys/stat.h>
32943 struct stat buffer;
32944 int status;
32945 ...
32946 status = lstat("/modules/pass1", &buffer);

```

32947 APPLICATION USAGE

32948 None.

32949 RATIONALE

32950 The intent of the paragraph describing “additional or alternate file access control mechanisms”
32951 is to allow a secure implementation where a process with a label that does not dominate the
32952 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with
32953 a label that dominates the file’s label does not need read permission. An implementation that
32954 supports write-up operations could fail *fstat()* function calls even though it has a valid file
32955 descriptor open for writing.

32956 The purpose of the *fstatat()* function is to obtain the status of files in directories other than the
32957 current working directory without exposure to race conditions. Any part of the path of a file
32958 could be changed in parallel to a call to *stat()*, resulting in unspecified behavior. By opening a

32959 file descriptor for the target directory and using the *fstatat()* function it can be guaranteed that
 32960 the file for which status is returned is located relative to the desired directory.

32961 FUTURE DIRECTIONS

32962 None.

32963 SEE ALSO

32964 *access()*, *chmod()*, *fdopendir()*, *fstat()*, *mknod()*, *readlink()*, *symlink()*

32965 XBD Section 4.9 (on page 109), [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)

32966 CHANGE HISTORY

32967 First released in Issue 1. Derived from Issue 1 of the SVID.

32968 Issue 5

32969 Large File Summit extensions are added.

32970 Issue 6

32971 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

32972 The following new requirements on POSIX implementations derive from alignment with the
 32973 Single UNIX Specification:

32974 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 32975 required for conforming implementations of previous POSIX specifications, it was not
 32976 required for UNIX applications.

32977 The [EIO] mandatory error condition is added.

32978 The [ELOOP] mandatory error condition is added.

32979 The [EOVERFLOW] mandatory error condition is added. This change is to support large
 32980 files.

32981 The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are
 32982 added.

32983 The following changes were made to align with the IEEE P1003.1a draft standard:

32984 Details are added regarding the treatment of symbolic links.

32985 The [ELOOP] optional error condition is added.

32986 The normative text is updated to avoid use of the term “must” for application requirements.

32987 The **restrict** keyword is added to the *stat()* prototype for alignment with the ISO/IEC 9899:1999
 32988 standard.

32989 Issue 7

32990 Austin Group Interpretation 1003.1-2001 #143 is applied.

32991 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify to which file types *st_nlink*
 32992 applies.

32993 The *fstatat()* function is added from The Open Group Technical Standard, 2006, Extended API
 32994 Set Part 2.

32995 Changes are made related to support for finegrained timestamps.

32996 The *lstat()* function is now required to return meaningful data for symbolic links in all **stat**
 32997 structure fields, except for the permission bits of *st_mode*.

32998 Changes are made to allow a directory to be opened for searching.

32999 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
33000 pathname exists but is not a directory or a symbolic link to a directory.

33001 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0198 [461], XSH/TC1-2008/0199 [324],
33002 XSH/TC1-2008/0200 [278], XSH/TC1-2008/0201 [278], and XSH/TC1-2008/0202 [291] are
33003 applied.

33004 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0136 [591], XSH/TC2-2008/0137 [817],
33005 XSH/TC2-2008/0138 [817], and XSH/TC2-2008/0139 [889] are applied.

33006 **NAME**

33007 fstatvfs, statvfs ‡'get file system information

33008 **SYNOPSIS**

33009 #include <sys/statvfs.h>

33010 int fstatvfs(int *fildev*, struct statvfs **buf*);33011 int statvfs(const char *restrict *path*, struct statvfs *restrict *buf*);33012 **DESCRIPTION**33013 The *fstatvfs()* function shall obtain information about the file system containing the file
33014 referenced by *fildev*.33015 The *statvfs()* function shall obtain information about the file system containing the file named by
33016 *path*.33017 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,
33018 write, or execute permission of the named file is not required.33019 The following flags can be returned in the *f_flag* member:

33020 ST_RDONLY Read-only file system.

33021 ST_NOSUID Setuid/setgid bits ignored by *exec*.33022 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file
33023 systems.33024 **RETURN VALUE**33025 Upon successful completion, *statvfs()* shall return 0. Otherwise, it shall return -1 and set *errno* to
33026 indicate the error.33027 **ERRORS**33028 The *fstatvfs()* and *statvfs()* functions shall fail if:

33029 [EIO] An I/O error occurred while reading the file system.

33030 [EINTR] A signal was caught during execution of the function.

33031 [EOVERFLOW] One of the values to be returned cannot be represented correctly in the
33032 structure pointed to by *buf*.33033 The *fstatvfs()* function shall fail if:33034 [EBADF] The *fildev* argument is not an open file descriptor.33035 The *statvfs()* function shall fail if:

33036 [EACCES] Search permission is denied on a component of the path prefix.

33037 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
33038 argument.

33039 [ENAMETOOLONG]

33040 The length of a component of a pathname is longer than {NAME_MAX}.

33041 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.33042 [ENOTDIR] A component of the path prefix names an existing file that is neither a
33043 directory nor a symbolic link to a directory, or the *path* argument contains at
33044 least one non-*<slash>* character and ends with one or more trailing *<slash>*
33045 characters and the last pathname component names an existing file that is
33046 neither a directory nor a symbolic link to a directory.

33047 The *statvfs()* function may fail if:

33048 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
33049 resolution of the *path* argument.

33050 [ENAMETOOLONG]
33051 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
33052 symbolic link produced an intermediate result with a length that exceeds
33053 {PATH_MAX}.

33054 EXAMPLES

33055 Obtaining File System Information Using *fstatvfs()*

33056 The following example shows how to obtain file system information for the file system upon
33057 which the file named **/home/cnd/mod1** resides, using the *fstatvfs()* function. The
33058 **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed
33059 to the *fstatvfs()* function.

```
33060 #include <sys/statvfs.h>
33061 #include <fcntl.h>
33062
33063 struct statvfs buffer;
33064 int status;
33065 ...
33066 fildes = open("/home/cnd/mod1", O_RDWR);
33067 status = fstatvfs(fildes, &buffer);
```

33067 Obtaining File System Information Using *statvfs()*

33068 The following example shows how to obtain file system information for the file system upon
33069 which the file named **/home/cnd/mod1** resides, using the *statvfs()* function.

```
33070 #include <sys/statvfs.h>
33071
33072 struct statvfs buffer;
33073 int status;
33074 ...
33075 status = statvfs("/home/cnd/mod1", &buffer);
```

33075 APPLICATION USAGE

33076 None.

33077 RATIONALE

33078 None.

33079 FUTURE DIRECTIONS

33080 None.

33081 SEE ALSO

33082 *chmod()*, *chown()*, *creat()*, *dup()*, *exec*, *fcntl()*, *link()*, *mknod()*, *open()*, *pipe()*, *read()*, *time()*,
33083 *unlink()*, *utime()*, *write()*

33084 XBD [<sys/statvfs.h>](#)

33085 **CHANGE HISTORY**

33086 First released in Issue 4, Version 2.

33087 **Issue 5**

33088 Moved from X/OPEN UNIX extension to BASE.

33089 Large File Summit extensions are added.

33090 **Issue 6**

33091 The normative text is updated to avoid use of the term “must” for application requirements.

33092 The **restrict** keyword is added to the *statvfs()* prototype for alignment with the
33093 ISO/IEC 9899:1999 standard.

33094 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
33095 [ELOOP] error condition is added.

33096 **Issue 7**

33097 Austin Group Interpretation 1003.1-2001 #143 is applied.

33098 SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.

33099 The *fstatvfs()* and *statvfs()* functions are moved from the XSI option to the Base.

33100 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
33101 pathname exists but is not a directory or a symbolic link to a directory.

33102 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0203 [324] is applied.

33103 **NAME**

33104 fsync — synchronize changes to a file

33105 **SYNOPSIS**

```
33106 #include <unistd.h>
33107 int fsync(int fildev);
```

33108 **DESCRIPTION**

33109 The *fsync()* function shall request that all data for the open file descriptor named by *fildev* is to be
 33110 transferred to the storage device associated with the file described by *fildev*. The nature of the
 33111 transfer is implementation-defined. The *fsync()* function shall not return until the system has
 33112 completed that action or until an error is detected.

33113 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued
 33114 I/O operations associated with the file indicated by file descriptor *fildev* to the synchronized I/O
 33115 completion state. All I/O operations shall be completed as defined for synchronized I/O file
 33116 integrity completion.

33117 **RETURN VALUE**

33118 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set
 33119 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed
 33120 to have been completed.

33121 **ERRORS**

33122 The *fsync()* function shall fail if:

- 33123 [EBADF] The *fildev* argument is not a valid descriptor.
- 33124 [EINTR] The *fsync()* function was interrupted by a signal.
- 33125 [EINVAL] The *fildev* argument does not refer to a file on which this operation is possible.
- 33126 [EIO] An I/O error occurred while reading from or writing to the file system.

33127 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions
 33128 defined for *read()* and *write()*.

33129 **EXAMPLES**

33130 None.

33131 **APPLICATION USAGE**

33132 The *fsync()* function should be used by programs which require modifications to a file to be
 33133 completed before continuing; for example, a program which contains a simple transaction
 33134 facility might use it to ensure that all modifications to a file or files caused by a transaction are
 33135 recorded.

33136 **RATIONALE**

33137 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to
 33138 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is
 33139 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and
 33140 “non-volatile storage” are not defined here, the wording has to be more abstract.

33141 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance
 33142 document to tell the user what can be expected from the system. It is explicitly intended that a
 33143 null implementation is permitted. This could be valid in the case where the system cannot assure
 33144 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the
 33145 functionality is not required. In the middle ground between these extremes, *fsync()* might or
 33146 might not actually cause data to be written where it is safe from a power failure. The

33147 conformance document should identify at least that one configuration exists (and how to obtain
33148 that configuration) where this can be assured for at least some files that the user can select to use
33149 for critical data. It is not intended that an exhaustive list is required, but rather sufficient
33150 information is provided so that if critical data needs to be saved, the user can determine how the
33151 system is to be configured to allow the data to be written to non-volatile storage.

33152 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.
33153 That does not make the function any less valuable, just more difficult to test. A formal
33154 conformance test should probably force a system crash (power shutdown) during the test for
33155 this condition, but it needs to be done in such a way that automated testing does not require this
33156 to be done except when a formal record of the results is being made. It would also not be
33157 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation
33158 issue.

33159 **FUTURE DIRECTIONS**

33160 None.

33161 **SEE ALSO**

33162 [*sync\(\)*](#)

33163 XBD [`<unistd.h>`](#)

33164 **CHANGE HISTORY**

33165 First released in Issue 3.

33166 **Issue 5**

33167 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and
33168 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate
33169 that *fsync()* can return the error conditions defined for *read()* and *write()*.

33170 **Issue 6**

33171 This function is marked as part of the File Synchronization option.

33172 The following new requirements on POSIX implementations derive from alignment with the
33173 Single UNIX Specification:

33174 The [EINVAL] and [EIO] mandatory error conditions are added.

33175 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/44 is applied, applying an editorial
33176 rewording of the DESCRIPTION. No change in meaning is intended.

33177 **NAME**

33178 ftell, ftello — return a file offset in a stream

33179 **SYNOPSIS**

33180 #include <stdio.h>

33181 long ftell(FILE *stream);

33182 CX off_t ftello(FILE *stream);

33183 **DESCRIPTION**33184 CX The functionality described on this reference page is aligned with the ISO C standard. Any
33185 conflict between the requirements described here and the ISO C standard is unintentional. This
33186 volume of POSIX.1-2017 defers to the ISO C standard.33187 The *ftell()* function shall obtain the current value of the file-position indicator for the stream
33188 pointed to by *stream*.33189 The *ftell()* function shall not change the setting of *errno* if successful.33190 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off_t** and
33191 the *ftello()* function may change the setting of *errno* if successful.33192 **RETURN VALUE**33193 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position
33194 indicator for the stream measured in bytes from the beginning of the file.33195 Otherwise, *ftell()* and *ftello()* shall return -1 , and set *errno* to indicate the error.33196 **ERRORS**33197 CX The *ftell()* and *ftello()* functions shall fail if:33198 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.33199 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of
33200 type **long**.33201 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object
33202 of type **off_t**.33203 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.33204 **EXAMPLES**

33205 None.

33206 **APPLICATION USAGE**

33207 None.

33208 **RATIONALE**

33209 None.

33210 **FUTURE DIRECTIONS**

33211 None.

33212 **SEE ALSO**33213 [Section 2.5](#) (on page 495), [fgetpos\(\)](#), [fopen\(\)](#), [fseek\(\)](#), [lseek\(\)](#)33214 XBD [<stdio.h>](#)

33215 **CHANGE HISTORY**

33216 First released in Issue 1. Derived from Issue 1 of the SVID.

33217 **Issue 5**

33218 Large File Summit extensions are added.

33219 **Issue 6**

33220 Extensions beyond the ISO C standard are marked.

33221 The following new requirements on POSIX implementations derive from alignment with the
33222 Single UNIX Specification:

33223 The *ftello()* function is added.

33224 The [Eoverflow] error conditions are added.

33225 An additional [ESPIPE] error condition is added for sockets.

33226 **Issue 7**

33227 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0204 [105], XSH/TC1-2008/0205 [421],
33228 XSH/TC1-2008/0206 [122], XSH/TC1-2008/0207 [122], and XSH/TC1-2008/0208 [14] are
33229 applied.

33230 **NAME**

33231 ftok ‡generate an IPC key

33232 **SYNOPSIS**

```
33233 XSI #include <sys/ipc.h>
33234 key_t ftok(const char *path, int id);
```

33235 **DESCRIPTION**

33236 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to
 33237 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the
 33238 pathname of an existing file that the process is able to *stat()*, with the exception that if *stat()*
 33239 would fail with [EOVERFLOW] due to file size, *ftok()* shall still succeed.

33240 The *ftok()* function shall return the same key value for all paths that name the same file, when
 33241 called with the same *id* value, and should return different key values when called with different
 33242 *id* values or with paths that name different files existing on the same file system at the same
 33243 time. It is unspecified whether *ftok()* shall return the same key value when called again after the
 33244 file named by *path* is removed and recreated with the same name.

33245 Only the low-order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits
 33246 are 0.

33247 **RETURN VALUE**

33248 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key_t**)-1
 33249 and set *errno* to indicate the error.

33250 **ERRORS**

33251 The *ftok()* function shall fail if:

- | | | |
|-------|----------------|---|
| 33252 | [EACCES] | Search permission is denied for a component of the path prefix. |
| 33253 | [EIO] | An error occurred while reading from the file system. |
| 33254 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i>
33255 argument. |
| 33256 | [ENAMETOOLONG] | |
| 33257 | | The length of a component of a pathname is longer than {NAME_MAX}. |
| 33258 | [ENOENT] | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string. |
| 33259 | [ENOTDIR] | A component of the path prefix names an existing file that is neither a
33260 directory nor a symbolic link to a directory, or the <i>path</i> argument contains at
33261 least one non- <i><slash></i> character and ends with one or more trailing <i><slash></i>
33262 characters and the last pathname component names an existing file that is
33263 neither a directory nor a symbolic link to a directory. |

33264 The *ftok()* function may fail if:

- | | | |
|-------|----------------|---|
| 33265 | [ELOOP] | More than {SYMLOOP_MAX} symbolic links were encountered during
33266 resolution of the <i>path</i> argument. |
| 33267 | [ENAMETOOLONG] | |
| 33268 | | The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
33269 symbolic link produced an intermediate result with a length that exceeds
33270 {PATH_MAX}. |

33271 **EXAMPLES**33272 **Getting an IPC Key**

33273 The following example gets a key based on the pathname `/tmp` and the ID value `a`. It also
 33274 assigns the value of the resulting key to the `semkey` variable so that it will be available to a later
 33275 call to `semget()`, `msgget()`, or `shmget()`.

```
33276 #include <sys/ipc.h>
33277 ...
33278 key_t semkey;
33279 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
33280     perror("IPC error: ftok"); exit(1);
33281 }
```

33282 **APPLICATION USAGE**

33283 For maximum portability, `id` should be a single-byte character.

33284 Applications should not assume that the resulting key value is unique.

33285 **RATIONALE**

33286 None.

33287 **FUTURE DIRECTIONS**

33288 Future versions of this standard may add new interfaces to provide unique keys.

33289 **SEE ALSO**

33290 [*msgget\(\)*](#), [*semget\(\)*](#), [*shmget\(\)*](#)

33291 XBD [**<sys/ipc.h>**](#)

33292 **CHANGE HISTORY**

33293 First released in Issue 4, Version 2.

33294 **Issue 5**

33295 Moved from X/OPEN UNIX extension to BASE.

33296 **Issue 6**

33297 The normative text is updated to avoid use of the term “must” for application requirements.

33298 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 33299 [ELOOP] error condition is added.

33300 **Issue 7**

33301 Austin Group Interpretation 1003.1-2001 #143 is applied.

33302 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
 33303 pathname exists but is not a directory or a symbolic link to a directory.

33304 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0209 [343], XSH/TC1-2008/0210 [366],
 33305 XSH/TC1-2008/0211 [343], XSH/TC1-2008/0212 [324], XSH/TC1-2008/0213 [366],
 33306 XSH/TC1-2008/0214 [366], XSH/TC1-2008/0215 [366], and XSH/TC1-2008/0216 [366] are
 33307 applied.

33308 **NAME**

33309 ftruncate — truncate a file to a specified length

33310 **SYNOPSIS**

33311 #include <unistd.h>

33312 int ftruncate(int *fildev*, off_t *length*);33313 **DESCRIPTION**33314 If *fildev* is not a valid file descriptor open for writing, the *ftruncate()* function shall fail.

33315 If *fildev* refers to a regular file, the *ftruncate()* function shall cause the size of the file to be
 33316 truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no
 33317 longer be available to reads on the file. If the file previously was smaller than this size,
 33318 *ftruncate()* shall increase the size of the file. If the file size is increased, the extended area shall
 33319 appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to
 33320 *ftruncate()*.

33321 Upon successful completion, if *fildev* refers to a regular file, *ftruncate()* shall mark for update the
 33322 last data modification and last file status change timestamps of the file and the S_ISUID and
 33323 S_ISGID bits of the file mode may be cleared. If the *ftruncate()* function is unsuccessful, the file is
 33324 unaffected.

33325 XSI If the request would cause the file size to exceed the soft file size limit for the process, the
 33326 request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

33327 If *fildev* refers to a directory, *ftruncate()* shall fail.33328 If *fildev* refers to any other file type, except a shared memory object, the result is unspecified.

33329 SHM If *fildev* refers to a shared memory object, *ftruncate()* shall set the size of the shared memory
 33330 object to *length*.

33331 SHM If the effect of *ftruncate()* is to decrease the size of a memory mapped file or a shared memory
 33332 object and whole pages beyond the new end were previously mapped, then the whole pages
 33333 beyond the new end shall be discarded.

33334 References to discarded pages shall result in the generation of a SIGBUS signal.

33335 If the effect of *ftruncate()* is to increase the size of a memory object, it is unspecified whether the
 33336 contents of any mapped pages between the old end-of-file and the new are flushed to the
 33337 underlying object.

33338 **RETURN VALUE**

33339 Upon successful completion, *ftruncate()* shall return 0; otherwise, -1 shall be returned and *errno*
 33340 set to indicate the error.

33341 **ERRORS**33342 The *ftruncate()* function shall fail if:

33343 [EINTR] A signal was caught during execution.

33344 [EINVAL] The *length* argument was less than 0.

33345 [EFBIG] or [EINVAL]

33346 The *length* argument was greater than the maximum file size.

33347 [EFBIG] The file is a regular file and *length* is greater than the offset maximum
 33348 established in the open file description associated with *fildev*.

33349 [EIO] An I/O error occurred while reading from or writing to a file system.

33350 [EBADF] or [EINVAL]

33351 The *fildest* argument is not a file descriptor open for writing.

33352 EXAMPLES

33353 None.

33354 APPLICATION USAGE

33355 None.

33356 RATIONALE

33357 None.

33358 FUTURE DIRECTIONS

33359 None.

33360 SEE ALSO

33361 *open()*, *truncate()*

33362 XBD <*unistd.h*>

33363 CHANGE HISTORY

33364 First released in Issue 4, Version 2.

33365 Issue 5

33366 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX
33367 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is
33368 added to the list of mandatory errors that can be returned by *ftruncate()*.

33369 Large File Summit extensions are added.

33370 Issue 6

33371 The *truncate()* function is split out into a separate reference page.

33372 The following new requirements on POSIX implementations derive from alignment with the
33373 Single UNIX Specification:

33374 The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a
33375 regular file, the S_ISUID and S_ISGID bits in the file mode may be cleared.

33376 The following changes were made to align with the IEEE P1003.1a draft standard:

33377 The DESCRIPTION text is updated.

33378 XSI-conformant systems are required to increase the size of the file if the file was previously
33379 smaller than the size requested.

33380 Issue 7

33381 Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although
33382 the [EINVAL] “may fail” error was subsequently removed during review of the XSI option).

33383 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
33384 the Base.

33385 The DESCRIPTION is updated so that a call to *ftruncate()* when the file is smaller than the size
33386 requested will increase the size of the file. Previously, non-XSI-conforming implementations
33387 were allowed to increase the size of the file or fail.

33388 Changes are made related to support for finegrained timestamps.

33389 **NAME**

33390 ftrylockfile ‡stdio locking functions

33391 **SYNOPSIS**

```
33392 CX #include <stdio.h>  
33393 int ftrylockfile(FILE *file);
```

33394 **DESCRIPTION**

33395 Refer to *flockfile()*.

33396 **NAME**

33397 ftw — traverse (walk) a file tree

33398 **SYNOPSIS**

```
33399 OB XSI #include <ftw.h>
33400 int ftw(const char *path, int (*fn)(const char *,
33401         const struct stat *ptr, int flag), int ndirs);
```

33402 **DESCRIPTION**

33403 The *ftw()* function shall recursively descend the directory hierarchy rooted in *path*. For each
 33404 object in the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a null-
 33405 terminated character string containing the name of the object, a pointer to a **stat** structure
 33406 containing information about the object, filled in as if *stat()* or *lstat()* had been called to retrieve
 33407 the information. Possible values of the integer, defined in the **<ftw.h>** header, are:

33408 FTW_D For a directory.

33409 FTW_DNR For a directory that cannot be read.

33410 FTW_F For a non-directory file.

33411 FTW_SL For a symbolic link (but see also FTW_NS below).

33412 FTW_NS For an object other than a symbolic link on which *stat()* could not successfully be
 33413 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether
 33414 *ftw()* passes FTW_SL or FTW_NS to the user-supplied function.

33415 If the integer is FTW_DNR, descendants of that directory shall not be processed. If the integer is
 33416 FTW_NS, the **stat** structure contains undefined values. An example of an object that would
 33417 cause FTW_NS to be passed to the function pointed to by *fn* would be a file in a directory with
 33418 read but without execute (search) permission.

33419 The *ftw()* function shall visit a directory before visiting any of its descendants.33420 The *ftw()* function shall use at most one file descriptor for each level in the tree.33421 The argument *ndirs* should be in the range [1,{OPEN_MAX}].

33422 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a
 33423 non-zero value, or some error, other than [EACCES], is detected within *ftw()*.

33424 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors
 33425 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any
 33426 directory streams and file descriptors it uses not counting any opened by the application-
 33427 supplied *fn* function.

33428 The results are unspecified if the application-supplied *fn* function does not preserve the current
 33429 working directory.

33430 The *ftw()* function need not be thread-safe.33431 **RETURN VALUE**

33432 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero
 33433 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function
 33434 pointed to by *fn*. If *ftw()* detects an error, it shall return -1 and set *errno* to indicate the error.

33435 If *ftw()* encounters an error other than [EACCES] (see FTW_DNR and FTW_NS above), it shall
 33436 return -1 and set *errno* to indicate the error. The external variable *errno* may contain any error
 33437 value that is possible when a directory is opened or when one of the *stat* functions is executed on

33438 a directory or file.

33439 ERRORS

33440 The *ftw()* function shall fail if:

33441 [EACCES] Search permission is denied for any component of *path* or read permission is
33442 denied for *path*.

33443 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
33444 argument.

33445 [ENAMETOOLONG]

33446 The length of a component of a pathname is longer than {NAME_MAX}.

33447 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

33448 [ENOTDIR] A component of *path* names an existing file that is neither a directory nor a
33449 symbolic link to a directory.

33450 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
33451 programming environment for one or more files found in the file hierarchy.

33452 The *ftw()* function may fail if:

33453 [EINVAL] The value of the *ndirs* argument is invalid.

33454 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
33455 resolution of the *path* argument.

33456 [ENAMETOOLONG]

33457 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
33458 symbolic link produced an intermediate result with a length that exceeds
33459 {PATH_MAX}.

33460 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set
33461 accordingly.

33462 EXAMPLES

33463 Walking a Directory Structure

33464 The following example walks the current directory structure, calling the *fn* function for every
33465 directory entry, using at most 10 file descriptors:

```
33466 #include <ftw.h>
33467 ...
33468 if (ftw(".", fn, 10) != 0) {
33469     perror("ftw"); exit(2);
33470 }
```

33471 APPLICATION USAGE

33472 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly
33473 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*
33474 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains
33475 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has
33476 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next
33477 invocation.

33478 Applications should use the *nftw()* function instead of the obsolescent *ftw()* function.

33479 **RATIONALE**

33480 None.

33481 **FUTURE DIRECTIONS**33482 The *ftw()* function may be removed in a future version.33483 **SEE ALSO**33484 *fdopendir()*, *fstatat()*, *longjmp()*, *nftw()*, *siglongjmp()*33485 XBD <*ftw.h*>, <*sys/stat.h*>33486 **CHANGE HISTORY**

33487 First released in Issue 1. Derived from Issue 1 of the SVID.

33488 **Issue 5**

33489 UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

33490 **Issue 6**

33491 The ERRORS section is updated as follows:

33492 The wording of the mandatory [ELOOP] error condition is updated.

33493 A second optional [ELOOP] error condition is added.

33494 The [EOVERFLOW] mandatory error condition is added.

33495 A note is added to the DESCRIPTION indicating that this function need not be reentrant, and
33496 that the results are unspecified if the application-supplied *fn* function does not preserve the
33497 current working directory.33498 **Issue 7**

33499 Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

33500 SD5-XBD-ERN-61 is applied.

33501 The *ftw()* function is marked obsolescent.33502 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0217 [403], XSH/TC1-2008/0218 [324],
33503 and XSH/TC1-2008/0219 [361] are applied.

33504 **NAME**

33505 funlockfile ‡'stdio locking functions

33506 **SYNOPSIS**

```
33507 CX #include <stdio.h>  
33508 void funlockfile(FILE *file);
```

33509 **DESCRIPTION**

33510 Refer to *flockfile()*.

33511 **NAME**

33512 futimens, utimensat, utimes ‡set file access and modification times

33513 **SYNOPSIS**

33514 #include <sys/stat.h>

33515 int futimens(int *fd*, const struct timespec *times*[2]);

33516 OH #include <fcntl.h>

33517 int utimensat(int *fd*, const char **path*, const struct timespec *times*[2],
33518 int *flag*);

33519 XSI #include <sys/time.h>

33520 int utimes(const char **path*, const struct timeval *times*[2]);33521 **DESCRIPTION**

33522 The *futimens()* and *utimensat()* functions shall set the access and modification times of a file to
 33523 the values of the *times* argument. The *futimens()* function changes the times of the file associated
 33524 with the file descriptor *fd*. The *utimensat()* function changes the times of the file pointed to by
 33525 the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions
 33526 allow time specifications accurate to the nanosecond.

33527 For *futimens()* and *utimensat()*, the *times* argument is an array of two **timespec** structures. The
 33528 first array member represents the date and time of last access, and the second member
 33529 represents the date and time of last modification. The times in the **timespec** structure are
 33530 measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set
 33531 to the greatest value supported by the file system that is not greater than the specified time.

33532 If the *tv_nsec* field of a **timespec** structure has the special value **UTIME_NOW**, the file's relevant
 33533 timestamp shall be set to the greatest value supported by the file system that is not greater than
 33534 the current time. If the *tv_nsec* field has the special value **UTIME_OMIT**, the file's relevant
 33535 timestamp shall not be changed. In either case, the *tv_sec* field shall be ignored.

33536 If the *times* argument is a null pointer, both the access and modification timestamps shall be set
 33537 to the greatest value supported by the file system that is not greater than the current time. If
 33538 *utimensat()* is passed a relative path in the *path* argument, the file to be used shall be relative to
 33539 the directory associated with the file descriptor *fd* instead of the current working directory. If the
 33540 access mode of the open file description associated with the file descriptor is not **O_SEARCH**,
 33541 the function shall check whether directory searches are permitted using the current permissions
 33542 of the directory underlying the file descriptor. If the access mode is **O_SEARCH**, the function
 33543 shall not perform the check.

33544 If *utimensat()* is passed the special value **AT_FDCWD** in the *fd* parameter, the current working
 33545 directory shall be used.

33546 Only a process with the effective user ID equal to the user ID of the file, or with write access to
 33547 the file, or with appropriate privileges may use *futimens()* or *utimensat()* with a null pointer as
 33548 the *times* argument or with both *tv_nsec* fields set to the special value **UTIME_NOW**. Only a
 33549 process with the effective user ID equal to the user ID of the file or with appropriate privileges
 33550 may use *futimens()* or *utimensat()* with a non-null *times* argument that does not have both
 33551 *tv_nsec* fields set to **UTIME_NOW** and does not have both *tv_nsec* fields set to **UTIME_OMIT**. If
 33552 both *tv_nsec* fields are set to **UTIME_OMIT**, no ownership or permissions check shall be
 33553 performed for the file, but other error conditions may still be detected (including **[EACCES]**
 33554 errors related to the path prefix).

33555 Values for the *flag* argument of *utimensat()* are constructed by a bitwise-inclusive OR of flags

33556 from the following list, defined in `<fcntl.h>`:

33557 `AT_SYMLINK_NOFOLLOW`

33558 If *path* names a symbolic link, then the access and modification times of the symbolic link

33559 are changed.

33560 Upon successful completion, *futimens()* and *utimensat()* shall mark the last file status change

33561 timestamp for update, with the exception that if both *tv_nsec* fields are set to `UTIME_OMIT`, the

33562 file status change timestamp need not be marked for update.

33563 The *utimes()* function shall be equivalent to the *utimensat()* function with the special value

33564 `AT_FDCWD` as the *fd* argument and the *flag* argument set to zero, except that the *times* argument

33565 is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond,

33566 not nanosecond, and rounding towards the nearest second may occur.

33567 **RETURN VALUE**

33568 Upon successful completion, these functions shall return 0. Otherwise, these functions shall

33569 return `-1` and set *errno* to indicate the error. If `-1` is returned, the file times shall not be affected.

33570 **ERRORS**

33571 These functions shall fail if:

33572 [EACCES] The *times* argument is a null pointer, or both *tv_nsec* values are `UTIME_NOW`,

33573 and the effective user ID of the process does not match the owner of the file

33574 and write access is denied.

33575 [EINVAL] Either of the *times* argument structures specified a *tv_nsec* value that was

33576 neither `UTIME_NOW` nor `UTIME_OMIT`, and was a value less than zero or

33577 greater than or equal to 1 000 million.

33578 [EINVAL] A new file timestamp would be a value whose *tv_sec* component is not a value

33579 supported by the file system.

33580 [EPERM] The *times* argument is not a null pointer, does not have both *tv_nsec* fields set

33581 to `UTIME_NOW`, does not have both *tv_nsec* fields set to `UTIME_OMIT`, the

33582 calling process' effective user ID does not match the owner of the file, and the

33583 calling process does not have appropriate privileges.

33584 [EROFS] The file system containing the file is read-only.

33585 The *futimens()* function shall fail if:

33586 [EBADF] The *fd* argument is not a valid file descriptor.

33587 The *utimensat()* function shall fail if:

33588 [EACCES] The access mode of the open file description associated with *fd* is not

33589 `O_SEARCH` and the permissions of the directory underlying *fd* do not permit

33590 directory searches.

33591 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is

33592 neither `AT_FDCWD` nor a valid file descriptor open for reading or searching.

33593 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated

33594 with a non-directory file.

33595 The *utimensat()* and *utimes()* functions shall fail if:

33596 [EACCES] Search permission is denied by a component of the path prefix.

- 33597 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
33598 argument.
- 33599 [ENAMETOOLONG]
33600 The length of a component of a pathname is longer than {NAME_MAX}.
- 33601 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 33602 [ENOTDIR] A component of the path prefix names an existing file that is neither a
33603 directory nor a symbolic link to a directory, or the *path* argument contains at
33604 least one non-`<slash>` character and ends with one or more trailing `<slash>`
33605 characters and the last pathname component names an existing file that is
33606 neither a directory nor a symbolic link to a directory.
- 33607 The *utimensat()* and *utimes()* functions may fail if:
- 33608 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
33609 resolution of the *path* argument.
- 33610 [ENAMETOOLONG]
33611 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
33612 symbolic link produced an intermediate result with a length that exceeds
33613 {PATH_MAX}.
- 33614 The *utimensat()* function may fail if:
- 33615 [EINVAL] The value of the *flag* argument is not valid.

33616 EXAMPLES

33617 None.

33618 APPLICATION USAGE

33619 None.

33620 RATIONALE

33621 The purpose of the *utimensat()* function is to set the access and modification time of files in
33622 directories other than the current working directory without exposure to race conditions. Any
33623 part of the path of a file could be changed in parallel to a call to *utimes()*, resulting in unspecified
33624 behavior. By opening a file descriptor for the target directory and using the *utimensat()* function
33625 it can be guaranteed that the changed file is located relative to the desired directory.

33626 The standard developers considered including a special case for the permissions required by
33627 *utimensat()* when one *tv_nsec* field is `UTIME_NOW` and the other is `UTIME_OMIT`. One
33628 possibility would be to include this case in with the cases where *times* is a null pointer or both
33629 fields are `UTIME_NOW`, where the call is allowed if the process has write permission for the file.
33630 However, associating write permission with an update to just the last data access timestamp
33631 (which is normally updated by *read()*) did not seem appropriate. The other possibility would be
33632 to specify that this one case is allowed if the process has read permission, but this was felt to be
33633 too great a departure from the *utime()* and *utimes()* functions on which *utimensat()* is based. If
33634 an application needs to set the last data access timestamp to the current time for a file on which
33635 it has read permission but is not the owner, it can do so by opening the file, reading one or more
33636 bytes (or reading a directory entry, if the file is a directory), and then closing it.

33637 FUTURE DIRECTIONS

33638 None.

33639 **SEE ALSO**33640 *read()*, *utime()*33641 XBD [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/time.h>](#)33642 **CHANGE HISTORY**

33643 First released in Issue 4, Version 2.

33644 **Issue 5**

33645 Moved from X/OPEN UNIX extension to BASE.

33646 **Issue 6**

33647 This function is marked LEGACY.

33648 The normative text is updated to avoid use of the term “must” for application requirements.

33649 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
33650 [ELOOP] error condition is added.33651 **Issue 7**

33652 Austin Group Interpretation 1003.1-2001 #143 is applied.

33653 The LEGACY marking is removed.

33654 The *utimensat()* function (renamed from *futimesat()*) is added from The Open Group Technical
33655 Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by
33656 adding a *flag* argument.33657 The *futimens()* function is added.

33658 Changes are made related to support for finegrained timestamps.

33659 Changes are made to allow a directory to be opened for searching.

33660 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
33661 pathname exists but is not a directory or a symbolic link to a directory.33662 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0220 [63,428], XSH/TC1-2008/0221
33663 [278], XSH/TC1-2008/0222 [324], XSH/TC1-2008/0223 [306], and XSH/TC1-2008/0224 [278] are
33664 applied.33665 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0140 [591], XSH/TC2-2008/0141 [817],
33666 XSH/TC2-2008/0142 [485], and XSH/TC2-2008/0143 [817] are applied.

33667 **NAME**

33668 fwide — set stream orientation

33669 **SYNOPSIS**

33670 #include <stdio.h>

33671 #include <wchar.h>

33672 int fwide(FILE *stream, int mode);

33673 **DESCRIPTION**

33674 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 33675 conflict between the requirements described here and the ISO C standard is unintentional. This
 33676 volume of POSIX.1-2017 defers to the ISO C standard.

33677 The *fwide()* function shall determine the orientation of the stream pointed to by *stream*. If *mode* is
 33678 greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less
 33679 than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero
 33680 and the function does not alter the orientation of the stream.

33681 If the orientation of the stream has already been determined, *fwide()* shall not change it.

33682 CX The *fwide()* function shall not change the setting of *errno* if successful.

33683 Since no return value is reserved to indicate an error, an application wishing to check for error
 33684 situations should set *errno* to 0, then call *fwide()*, then check *errno*, and if it is non-zero, assume
 33685 an error has occurred.

33686 **RETURN VALUE**

33687 The *fwide()* function shall return a value greater than zero if, after the call, the stream has wide-
 33688 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no
 33689 orientation.

33690 **ERRORS**

33691 The *fwide()* function may fail if:

33692 CX [EBADF] The *stream* argument is not a valid stream.

33693 **EXAMPLES**

33694 None.

33695 **APPLICATION USAGE**

33696 A call to *fwide()* with *mode* set to zero can be used to determine the current orientation of a
 33697 stream.

33698 **RATIONALE**

33699 None.

33700 **FUTURE DIRECTIONS**

33701 None.

33702 **SEE ALSO**

33703 XBD <stdio.h>, <wchar.h>

33704 **CHANGE HISTORY**

33705 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 33706 (E).

33707 Issue 6

33708 Extensions beyond the ISO C standard are marked.

33709 Issue 7

33710 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0225 [272] is applied.

33711 **NAME**33712 `fwprintf`, `swprintf`, `wprintf` `‡`print formatted wide-character output33713 **SYNOPSIS**33714 `#include <stdio.h>`33715 `#include <wchar.h>`33716 `int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);`33717 `int swprintf(wchar_t *restrict ws, size_t n,`33718 `const wchar_t *restrict format, ...);`33719 `int wprintf(const wchar_t *restrict format, ...);`33720 **DESCRIPTION**

33721 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 33722 conflict between the requirements described here and the ISO C standard is unintentional. This
 33723 volume of POSIX.1-2017 defers to the ISO C standard.

33724 The `fwprintf()` function shall place output on the named output *stream*. The `wprintf()` function
 33725 shall place output on the standard output stream *stdout*. The `swprintf()` function shall place
 33726 output followed by the null wide character in consecutive wide characters starting at *ws*; no
 33727 more than *n* wide characters shall be written, including a terminating null wide character, which
 33728 is always added (unless *n* is zero).

33729 Each of these functions shall convert, format, and print its arguments under control of the *format*
 33730 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*,
 33731 which are simply copied to the output stream, and *conversion specifications*, each of which results
 33732 in the fetching of zero or more arguments. The results are undefined if there are insufficient
 33733 arguments for the *format*. If the *format* is exhausted while arguments remain, the excess
 33734 arguments are evaluated but are otherwise ignored.

33735 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 33736 to the next unused argument. In this case, the conversion specifier wide character `%` (see below)
 33737 is replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range
 33738 `[1,{NL_ARGMAX}]`, giving the position of the argument in the argument list. This feature
 33739 provides for the definition of *format* wide-character strings that select arguments in an order
 33740 appropriate to specific languages (see the EXAMPLES section).

33741 The *format* can contain either numbered argument specifications (that is, `"%n$"` and `"*m$"`), or
 33742 unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The only
 33743 exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing numbered
 33744 and unnumbered argument specifications in a *format* wide-character string are undefined. When
 33745 numbered argument specifications are used, specifying the *N*th argument requires that all the
 33746 leading arguments, from the first to the (*N*-1)th, are specified in the *format* wide-character string.

33747 In *format* wide-character strings containing the `"%n$"` form of conversion specification,
 33748 numbered arguments in the argument list can be referenced from the *format* wide-character
 33749 string as many times as required.

33750 In *format* wide-character strings containing the `%` form of conversion specification, each
 33751 argument in the argument list shall be used exactly once. It is unspecified whether an encoding
 33752 error occurs if the format string contains `wchar_t` values that do not correspond to members of
 33753 the character set of the current locale and the specified semantics do not require that value to be
 33754 processed by `wcrtomb()`.

33755 CX All forms of the `fwprintf()` function allow for the insertion of a locale-dependent radix character
 33756 in the output string, output as a wide-character value. The radix character is defined in the
 33757 current locale (category `LC_NUMERIC`). In the POSIX locale, or in a locale where the radix

- 33758 character is not defined, the radix character shall default to a <period> ('.').
- 33759 CX Each conversion specification is introduced by the '%' wide character or by the wide-character
33760 sequence "%n\$", after which the following appear in sequence:
- 33761 Zero or more *flags* (in any order), which modify the meaning of the conversion
33762 specification.
- 33763 An optional minimum *field width*. If the converted value has fewer wide characters than
33764 the field width, it shall be padded with <space> characters by default on the left; it shall be
33765 padded on the right, if the left-adjustment flag ('-'), described below, is given to the field
33766 width. The field width takes the form of an <asterisk> ('*'), described below, or a decimal
33767 integer.
- 33768 An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,
33769 x, and X conversion specifiers; the number of digits to appear after the radix character for
33770 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for
33771 the g and G conversion specifiers; or the maximum number of wide characters to be
33772 printed from a string in the s conversion specifiers. The precision takes the form of a
33773 <period> ('.') followed either by an <asterisk> ('*'), described below, or an optional
33774 decimal digit string, where a null digit string is treated as 0. If a precision appears with any
33775 other conversion wide character, the behavior is undefined.
- 33776 An optional length modifier that specifies the size of the argument.
- 33777 A *conversion specifier* wide character that indicates the type of conversion to be applied.
- 33778 A field width, or precision, or both, may be indicated by an <asterisk> ('*'). In this case an
33779 argument of type **int** supplies the field width or precision. Applications shall ensure that
33780 arguments specifying field width, or precision, or both appear in that order before the argument,
33781 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field
33782 CX width. A negative precision is taken as if the precision were omitted. In *format* wide-character
33783 strings containing the "%n\$" form of a conversion specification, a field width or precision may
33784 be indicated by the sequence "*m\$", where *m* is a decimal integer in the range
33785 [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an
33786 integer argument containing the field width or precision, for example:
- 33787 `wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);`
- 33788 The flag wide characters and their meanings are:
- 33789 CX ' (The <apostrophe>.) The integer portion of the result of a decimal conversion (%i, %d,
33790 %u, %f, %F, %g, or %G) shall be formatted with thousands' grouping wide characters. For
33791 other conversions, the behavior is undefined. The numeric grouping wide character is
33792 used.
- 33793 - The result of the conversion shall be left-justified within the field. The conversion shall
33794 be right-justified if this flag is not specified.
- 33795 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The
33796 conversion shall begin with a sign only when a negative value is converted if this flag is
33797 not specified.
- 33798 <space> If the first wide character of a signed conversion is not a sign, or if a signed conversion
33799 results in no wide characters, a <space> shall be prefixed to the result. This means that
33800 if the <space> and '+' flags both appear, the <space> flag shall be ignored.

33801	#	Specifies that the value is to be converted to an alternative form. For <code>o</code> conversion, it shall increase the precision, if and only if necessary, to force the first digit of the result to be zero (if the value and precision are both 0, a single 0 is printed). For <code>x</code> or <code>X</code> conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, the result shall always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For <code>g</code> and <code>G</code> conversion specifiers, trailing zeros shall <i>not</i> be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.
33802		
33803		
33804		
33805		
33806		
33807		
33808		
33809		
33810	0	For <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the <code>'0'</code> and <code>'-'</code> flags both appear, the <code>'0'</code> flag shall be ignored. For <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , and <code>X</code> conversion specifiers, if a precision is specified, the <code>'0'</code> flag shall be ignored. If the <code>'0'</code> and <code><apostrophe></code> flags both appear, the grouping wide characters are inserted before zero padding. For other conversions, the behavior is undefined.
33811		
33812		
33813		
33814	CX	
33815		
33816		
33817		The length modifiers and their meanings are:
33818	hh	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to signed char or unsigned char before printing); or that a following <code>n</code> conversion specifier applies to a pointer to a signed char argument.
33819		
33820		
33821		
33822		
33823	h	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a short or unsigned short argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to short or unsigned short before printing); or that a following <code>n</code> conversion specifier applies to a pointer to a short argument.
33824		
33825		
33826		
33827		
33828	l (ell)	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a long or unsigned long argument; that a following <code>n</code> conversion specifier applies to a pointer to a long argument; that a following <code>c</code> conversion specifier applies to a wint_t argument; that a following <code>s</code> conversion specifier applies to a pointer to a wchar_t argument; or has no effect on a following <code>a</code> , <code>A</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , or <code>G</code> conversion specifier.
33829		
33830		
33831		
33832		
33833	ll (ell-ell)	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a long long or unsigned long long argument; or that a following <code>n</code> conversion specifier applies to a pointer to a long long argument.
33834		
33835		
33836		
33837	j	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to an intmax_t or uintmax_t argument; or that a following <code>n</code> conversion specifier applies to a pointer to an intmax_t argument.
33838		
33839		
33840	z	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a size_t or the corresponding signed integer type argument; or that a following <code>n</code> conversion specifier applies to a pointer to a signed integer type corresponding to a size_t argument.
33841		
33842		
33843	t	Specifies that a following <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , or <code>X</code> conversion specifier applies to a ptrdiff_t or the corresponding unsigned type argument; or that a following <code>n</code> conversion specifier applies to a pointer to a ptrdiff_t argument.
33844		
33845		

- 33846 L Specifies that a following *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G* conversion specifier applies to a **long**
33847 **double** argument.
- 33848 If a length modifier appears with any conversion specifier other than as specified above, the
33849 behavior is undefined.
- 33850 The conversion specifiers and their meanings are:
- 33851 *d*, *i* The **int** argument shall be converted to a signed decimal in the style "*[-]dddd*". The
33852 precision specifies the minimum number of digits to appear; if the value being
33853 converted can be represented in fewer digits, it shall be expanded with leading zeros.
33854 The default precision shall be 1. The result of converting zero with an explicit precision
33855 of zero shall be no wide characters.
- 33856 *o* The **unsigned** argument shall be converted to unsigned octal format in the style
33857 "*dddd*". The precision specifies the minimum number of digits to appear; if the value
33858 being converted can be represented in fewer digits, it shall be expanded with leading
33859 zeros. The default precision shall be 1. The result of converting zero with an explicit
33860 precision of zero shall be no wide characters.
- 33861 *u* The **unsigned** argument shall be converted to unsigned decimal format in the style
33862 "*dddd*". The precision specifies the minimum number of digits to appear; if the value
33863 being converted can be represented in fewer digits, it shall be expanded with leading
33864 zeros. The default precision shall be 1. The result of converting zero with an explicit
33865 precision of zero shall be no wide characters.
- 33866 *x* The **unsigned** argument shall be converted to unsigned hexadecimal format in the style
33867 "*dddd*"; the letters "*abcdef*" are used. The precision specifies the minimum number
33868 of digits to appear; if the value being converted can be represented in fewer digits, it
33869 shall be expanded with leading zeros. The default precision shall be 1. The result of
33870 converting zero with an explicit precision of zero shall be no wide characters.
- 33871 *X* Equivalent to the *x* conversion specifier, except that letters "*ABCDEF*" are used instead
33872 of "*abcdef*".
- 33873 *f*, *F* The **double** argument shall be converted to decimal notation in the style
33874 "*[-]ddd.ddd*", where the number of digits after the radix character shall be equal to
33875 the precision specification. If the precision is missing, it shall be taken as 6; if the
33876 precision is explicitly zero and no '#' flag is present, no radix character shall appear. If
33877 a radix character appears, at least one digit shall appear before it. The value shall be
33878 rounded in an implementation-defined manner to the appropriate number of digits.
- 33879 A **double** argument representing an infinity shall be converted in one of the styles
33880 "*[-]inf*" or "*[-]infinity*"; which style is implementation-defined. A **double**
33881 argument representing a NaN shall be converted in one of the styles "*[-]nan*" or
33882 "*[-]nan(*n-char-sequence*)*"; which style, and the meaning of any *n-char-sequence*,
33883 is implementation-defined. The *F* conversion specifier produces "*INF*", "*INFINITY*",
33884 or "*NAN*" instead of "*inf*", "*infinity*", or "*nan*", respectively.
- 33885 *e*, *E* The **double** argument shall be converted in the style "*[-]d.ddde±dd*", where there
33886 shall be one digit before the radix character (which is non-zero if the argument is non-
33887 zero) and the number of digits after it shall be equal to the precision; if the precision is
33888 missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no
33889 radix character shall appear. The value shall be rounded in an implementation-defined
33890 manner to the appropriate number of digits. The *E* conversion wide character shall
33891 produce a number with '*E*' instead of '*e*' introducing the exponent. The exponent
33892 shall always contain at least two digits. If the value is zero, the exponent shall be zero.

33893 A **double** argument representing an infinity or NaN shall be converted in the style of
 33894 an `f` or `F` conversion specifier.

33895 `g, G` The **double** argument representing a floating-point number shall be converted in the
 33896 style `f` or `e` (or in the style `F` or `E` in the case of a `G` conversion specifier), depending on
 33897 the value converted and the precision. Let `P` equal the precision if non-zero, 6 if the
 33898 precision is omitted, or 1 if the precision is zero. Then, if a conversion with style `E`
 33899 would have an exponent of `X`:

33900 $\text{if } P \geq X \geq -4$, the conversion shall be with style `f` (or `F`) and precision `P-(X+1)`.
 33901 $\text{if } P < X$, otherwise, the conversion shall be with style `e` (or `E`) and precision `P-1`.

33902 Finally, unless the '#' flag is used, any trailing zeros shall be removed from the
 33903 fractional portion of the result and the decimal-point character shall be removed if there
 33904 is no fractional portion remaining.

33905 A **double** argument representing an infinity or NaN shall be converted in the style of
 33906 an `f` or `F` conversion specifier.

33907 `a, A` A **double** argument representing a floating-point number shall be converted in the
 33908 style "`[-]0xh.hhhhp±d`", where there shall be one hexadecimal digit (which is non-
 33909 zero if the argument is a normalized floating-point number and is otherwise
 33910 unspecified) before the decimal-point wide character and the number of hexadecimal
 33911 digits after it shall be equal to the precision; if the precision is missing and `FLT_RADIX`
 33912 is a power of 2, then the precision shall be sufficient for an exact representation of the
 33913 value; if the precision is missing and `FLT_RADIX` is not a power of 2, then the precision
 33914 shall be sufficient to distinguish values of type **double**, except that trailing zeros may
 33915 be omitted; if the precision is zero and the '#' flag is not specified, no decimal-point
 33916 wide character shall appear. The letters "abcdef" are used for a conversion and the
 33917 letters "ABCDEF" for A conversion. The A conversion specifier produces a number with
 33918 'X' and 'P' instead of 'x' and 'p'. The exponent shall always contain at least one
 33919 digit, and only as many more digits as necessary to represent the decimal exponent of
 33920 2. If the value is zero, the exponent shall be zero.

33921 A **double** argument representing an infinity or NaN shall be converted in the style of
 33922 an `f` or `F` conversion specifier.

33923 `c` If no `l` (ell) qualifier is present, the **int** argument shall be converted to a wide character
 33924 as if by calling the `btowc()` function and the resulting wide character shall be written.
 33925 Otherwise, the **wint_t** argument shall be converted to **wchar_t**, and written.

33926 `s` If no `l` (ell) qualifier is present, the application shall ensure that the argument is a
 33927 pointer to a character array containing a character sequence beginning in the initial
 33928 shift state. Characters from the array shall be converted as if by repeated calls to the
 33929 `mbrtowc()` function, with the conversion state described by an **mbstate_t** object
 33930 initialized to zero before the first character is converted, and written up to (but not
 33931 including) the terminating null wide character. If the precision is specified, no more
 33932 than that many wide characters shall be written. If the precision is not specified, or is
 33933 greater than the size of the array, the application shall ensure that the array contains a
 33934 null wide character.

33935 If an `l` (ell) qualifier is present, the application shall ensure that the argument is a
 33936 pointer to an array of type **wchar_t**. Wide characters from the array shall be written up
 33937 to (but not including) a terminating null wide character. If no precision is specified, or
 33938 is greater than the size of the array, the application shall ensure that the array contains a
 33939 null wide character. If a precision is specified, no more than that many wide characters

33940		shall be written.
33941	p	The application shall ensure that the argument is a pointer to void . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.
33942		
33943		
33944	n	The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <i>fwprintf()</i> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
33945		
33946		
33947		
33948		
33949	XSI	C Equivalent to <code>lc</code> .
33950	XSI	S Equivalent to <code>ls</code> .
33951	%	Output a ' <code>%</code> ' wide character; no argument shall be converted. The entire conversion specification shall be <code>%%</code> .
33952		
33953		If a conversion specification does not match one of the above forms, the behavior is undefined.
33954		In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by <i>fwprintf()</i> and <i>wprintf()</i> shall be printed as if <i>fputwc()</i> had been called.
33955		
33956		
33957		
33958		For <code>a</code> and <code>A</code> conversions, if <code>FLT_RADIX</code> is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.
33959		
33960		
33961		
33962		For <code>e</code> , <code>E</code> , <code>f</code> , <code>F</code> , <code>g</code> , and <code>G</code> conversion specifiers, if the number of significant decimal digits is at most <code>DECIMAL_DIG</code> , then the result should be correctly rounded. If the number of significant decimal digits is more than <code>DECIMAL_DIG</code> but the source value is exactly representable with <code>DECIMAL_DIG</code> digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having <code>DECIMAL_DIG</code> significant digits; the value of the resultant decimal string D should satisfy $L \leq D \leq U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.
33963		
33964		
33965		
33966		
33967		
33968		
33969		
33970	CX	The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful execution of <i>fwprintf()</i> or <i>wprintf()</i> and the next successful completion of a call to <i>fflush()</i> or <i>fclose()</i> on the same stream, or a call to <i>exit()</i> or <i>abort()</i> .
33971		
33972		
33973		
33974		RETURN VALUE
33975		Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of <i>swprintf()</i> , or a negative value if an output error was encountered, and set <i>errno</i> to indicate the error.
33976		
33977	CX	
33978		If <i>n</i> or more wide characters were requested to be written, <i>swprintf()</i> shall return a negative value, and set <i>errno</i> to indicate the error.
33979	CX	
33980		ERRORS
33981		For the conditions under which <i>fwprintf()</i> and <i>wprintf()</i> fail and may fail, refer to <i>fputwc()</i> .
33982		In addition, all forms of <i>fwprintf()</i> shall fail if:

33983 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been
 33984 detected.

33985 In addition, *fwprintf()* and *wprintf()* may fail if:

33986 CX [ENOMEM] Insufficient storage space is available.

33987 The *swprintf()* shall fail if:

33988 CX [EOVERFLOW] The value of *n* is greater than {INT_MAX} or the number of bytes needed to
 33989 hold the output excluding the terminating null is greater than {INT_MAX}.

33990 EXAMPLES

33991 To print the language-independent date and time format, the following statement could be used:

```
33992 wprintf(format, weekday, month, day, hour, min);
```

33993 For American usage, *format* could be a pointer to the wide-character string:

```
33994 L"%s, %s %d, %d:%.2d\n"
```

33995 producing the message:

```
33996 Sunday, July 3, 10:02
```

33997 whereas for German usage, *format* could be a pointer to the wide-character string:

```
33998 L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

33999 producing the message:

```
34000 Sonntag, 3. Juli, 10:02
```

34001 APPLICATION USAGE

34002 None.

34003 RATIONALE

34004 If an implementation detects that there are insufficient arguments for the format, it is
 34005 recommended that the function should fail and report an [EINVAL] error.

34006 FUTURE DIRECTIONS

34007 None.

34008 SEE ALSO

34009 [Section 2.5](#) (on page 495), [btowc\(\)](#), [fputwc\(\)](#), [fwscanf\(\)](#), [mbrtowc\(\)](#), [setlocale\(\)](#)

34010 [XBD Chapter 7](#) (on page 135), [<inttypes.h>](#), [<stdio.h>](#), [<wchar.h>](#)

34011 CHANGE HISTORY

34012 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 34013 (E).

34014 Issue 6

34015 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing
 34016 the case if *n* or more wide characters are requested to be written using *swprintf()*.

34017 The normative text is updated to avoid use of the term “must” for application requirements.

34018 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

34019 The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.

- 34020 The DESCRIPTION is updated.
- 34021 The hh, ll, j, t, and z length modifiers are added.
- 34022 The a, A, and F conversion characters are added.
- 34023 XSI shading is removed from the description of character string representations of infinity
34024 and NaN floating-point values.
- 34025 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
34026 specification” consistently.
- 34027 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.
- 34028 **Issue 7**
- 34029 Austin Group Interpretation 1003.1-2001 #161 is applied, updating the DESCRIPTION of the 0
34030 flag.
- 34031 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 34032 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied,
34033 revising the description of g and G.
- 34034 Functionality relating to the "%n\$" form of conversion specification and the <apostrophe> flag
34035 is moved from the XSI option to the Base.
- 34036 The [Eoverflow] error is added for *swprintf()*.
- 34037 Changes are made related to support for finegrained timestamps.
- 34038 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0226 [302] and XSH/TC1-2008/0227
34039 [14] are applied.
- 34040 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0144 [73], XSH/TC2-2008/0145 [894],
34041 XSH/TC2-2008/0146 [557], and XSH/TC2-2008/0147 [936] are applied.

34042 **NAME**34043 fwrite *†*binary output34044 **SYNOPSIS**

34045 #include <stdio.h>

```
34046 size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
34047 FILE *restrict stream);
```

34048 **DESCRIPTION**

34049 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 34050 conflict between the requirements described here and the ISO C standard is unintentional. This
 34051 volume of POSIX.1-2017 defers to the ISO C standard.

34052 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose
 34053 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be
 34054 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly
 34055 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by
 34056 the number of bytes successfully written. If an error occurs, the resulting value of the file-
 34057 position indicator for the stream is unspecified.

34058 CX The last data modification and last file status change timestamps of the file shall be marked for
 34059 update between the successful execution of *fwrite()* and the next successful completion of a call
 34060 to *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.

34061 **RETURN VALUE**

34062 The *fwrite()* function shall return the number of elements successfully written, which may be
 34063 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the
 34064 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for
 34065 CX the stream shall be set, and *errno* shall be set to indicate the error.

34066 **ERRORS**34067 Refer to *fputc()*.34068 **EXAMPLES**

34069 None.

34070 **APPLICATION USAGE**

34071 Because of possible differences in element length and byte ordering, files written using *fwrite()*
 34072 are application-dependent, and possibly cannot be read using *fread()* by a different application
 34073 or by the same application on a different processor.

34074 **RATIONALE**

34075 None.

34076 **FUTURE DIRECTIONS**

34077 None.

34078 **SEE ALSO**34079 [Section 2.5](#) (on page 495), *ferror()*, *fopen()*, *fprintf()*, *putc()*, *puts()*, *write()*34080 XBD [<stdio.h>](#)34081 **CHANGE HISTORY**

34082 First released in Issue 1. Derived from Issue 1 of the SVID.

34083 **Issue 6**

34084 Extensions beyond the ISO C standard are marked.

34085 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

34086 The *fwrite()* prototype is updated.34087 The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.34088 **Issue 7**

34089 Changes are made related to support for finegrained timestamps.

34090 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0228 [14] is applied.

34091 **NAME**

34092 fwscanf, swscanf, wscanf ‡convert formatted wide-character input

34093 **SYNOPSIS**

34094 #include <stdio.h>

34095 #include <wchar.h>

34096 int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ...);

34097 int swscanf(const wchar_t *restrict ws,

34098 const wchar_t *restrict format, ...);

34099 int wscanf(const wchar_t *restrict format, ...);

34100 **DESCRIPTION**

34101 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 34102 conflict between the requirements described here and the ISO C standard is unintentional. This
 34103 volume of POSIX.1-2017 defers to the ISO C standard.

34104 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read
 34105 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character
 34106 string *ws*. Each function reads wide characters, interprets them according to a format, and stores
 34107 the results in its arguments. Each expects, as arguments, a control wide-character string *format*
 34108 described below, and a set of *pointer* arguments indicating where the converted input should be
 34109 stored. The result is undefined if there are insufficient arguments for the format. If the *format* is
 34110 exhausted while arguments remain, the excess arguments are evaluated but are otherwise
 34111 ignored.

34112 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 34113 to the next unused argument. In this case, the conversion specifier wide character % (see below)
 34114 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range
 34115 [1,{NL_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that
 34116 select arguments in an order appropriate to specific languages. In *format* wide-character strings
 34117 containing the "%n\$" form of conversion specifications, it is unspecified whether numbered
 34118 arguments in the argument list can be referenced from the *format* wide-character string more
 34119 than once.

34120 The *format* can contain either form of a conversion specification ‡that is% or "%n\$" ‡úbut the
 34121 two forms cannot normally be mixed within a single *format* wide-character string. The only
 34122 exception to this is that %% or %* can be mixed with the "%n\$" form. When numbered argument
 34123 specifications are used, specifying the *N*th argument requires that all the leading arguments,
 34124 from the first to the (*N*-1)th, are pointers.

34125 The *fwscanf()* function in all its forms allows for detection of a language-dependent radix
 34126 character in the input string, encoded as a wide-character value. The radix character is defined
 34127 in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the
 34128 radix character is not defined, the radix character shall default to a <period> ('.').

34129 The *format* is a wide-character string composed of zero or more directives. Each directive is
 34130 composed of one of the following: one or more white-space wide characters (<space>, <tab>,
 34131 <newline>, <vertical-tab>, or <form-feed>); an ordinary wide character (neither '%' nor a
 34132 white-space character); or a conversion specification. It is unspecified whether an encoding
 34133 error occurs if the format string contains **wchar_t** values that do not correspond to members of
 34134 the character set of the current locale and the specified semantics do not require that value to be
 34135 processed by *wcrtomb()*.

34136 CX Each conversion specification is introduced by the '%' or by the character sequence "%n\$",
 34137 after which the following appear in sequence:

- 34138 An optional assignment-suppressing character '*'.
- 34139 An optional non-zero decimal integer that specifies the maximum field width.
- 34140 CX An optional assignment-allocation character 'm'.
- 34141 An optional length modifier that specifies the size of the receiving object.
- 34142 A conversion specifier wide character that specifies the type of conversion to be applied.
34143 The valid conversion specifiers are described below.
- 34144 The *fwscanf()* functions shall execute each directive of the format in turn. If a directive fails, as
34145 detailed below, the function shall return. Failures are described as input failures (due to the
34146 unavailability of input bytes) or matching failures (due to inappropriate input).
- 34147 A directive composed of one or more white-space wide characters is executed by reading input
34148 until no more valid input can be read, or up to the first wide character which is not a white-
34149 space wide character, which remains unread.
- 34150 A directive that is an ordinary wide character shall be executed as follows. The next wide
34151 character is read from the input and compared with the wide character that comprises the
34152 directive; if the comparison shows that they are not equivalent, the directive shall fail, and the
34153 differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding
34154 error, or a read error prevents a wide character from being read, the directive shall fail.
- 34155 A directive that is a conversion specification defines a set of matching input sequences, as
34156 described below for each conversion wide character. A conversion specification is executed in
34157 the following steps.
- 34158 Input white-space wide characters (as specified by *iswspace()*) shall be skipped, unless the
34159 conversion specification includes a [, c, or n conversion specifier.
- 34160 An item shall be read from the input, unless the conversion specification includes an n
34161 conversion specifier wide character. An input item is defined as the longest sequence of input
34162 wide characters, not exceeding any specified field width, which is an initial subsequence of a
34163 matching sequence. The first wide character, if any, after the input item shall remain unread. If
34164 the length of the input item is zero, the execution of the conversion specification shall fail; this
34165 condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented
34166 input from the stream, in which case it is an input failure.
- 34167 Except in the case of a % conversion specifier, the input item (or, in the case of a %n conversion
34168 specification, the count of input wide characters) shall be converted to a type appropriate to the
34169 conversion wide character. If the input item is not a matching sequence, the execution of the
34170 conversion specification shall fail; this condition is a matching failure. Unless assignment
34171 suppression was indicated by a '*', the result of the conversion shall be placed in the object
34172 pointed to by the first argument following the *format* argument that has not already received a
34173 CX conversion result if the conversion specification is introduced by %, or in the *n*th argument if
34174 introduced by the wide-character sequence "%n\$". If this object does not have an appropriate
34175 type, or if the result of the conversion cannot be represented in the space provided, the behavior
34176 is undefined.
- 34177 CX The %c, %s, and %[conversion specifiers shall accept an optional assignment-allocation
34178 character 'm', which shall cause a memory buffer to be allocated to hold the wide-character
34179 string converted including a terminating null wide character. In such a case, the argument
34180 corresponding to the conversion specifier should be a reference to a pointer value that will
34181 receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc()* had been
34182 called. The application shall be responsible for freeing the memory after usage. If there is
34183 insufficient memory to allocate a buffer, the function shall set *errno* to [ENOMEM] and a

34184 conversion error shall result. If the function returns EOF, any memory successfully allocated for
 34185 parameters using assignment-allocation character 'm' by this call shall be freed before the
 34186 function returns.

34187 The length modifiers and their meanings are:

34188 hh Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34189 argument with type pointer to **signed char** or **unsigned char**.

34190 h Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34191 argument with type pointer to **short** or **unsigned short**.

34192 l (ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34193 argument with type pointer to **long** or **unsigned long**; that a following a, A, e, E, f, F,
 34194 g, or G conversion specifier applies to an argument with type pointer to **double**; or that
 34195 a following c, s, or [conversion specifier applies to an argument with type pointer to
 34196 **wchar_t**. If the 'm' assignment-allocation character is specified, the conversion
 34197 applies to an argument with the type pointer to a pointer to **wchar_t**.

34198 ll (ell-ell) Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34199 argument with type pointer to **long long** or **unsigned long long**.
 34200

34201 j Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34202 argument with type pointer to **intmax_t** or **uintmax_t**.

34203 z Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34204 argument with type pointer to **size_t** or the corresponding signed integer type.

34205 t Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an
 34206 argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

34207 L Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an
 34208 argument with type pointer to **long double**.

34209 If a length modifier appears with any conversion specifier other than as specified above, the
 34210 behavior is undefined.

34211 The following conversion specifier wide characters are valid:

34212 d Matches an optionally signed decimal integer, whose format is the same as expected for
 34213 the subject sequence of *wcstol()* with the value 10 for the *base* argument. In the absence
 34214 of a size modifier, the application shall ensure that the corresponding argument is a
 34215 pointer to **int**.

34216 i Matches an optionally signed integer, whose format is the same as expected for the
 34217 subject sequence of *wcstol()* with 0 for the *base* argument. In the absence of a size
 34218 modifier, the application shall ensure that the corresponding argument is a pointer to
 34219 **int**.

34220 o Matches an optionally signed octal integer, whose format is the same as expected for
 34221 the subject sequence of *wcstoul()* with the value 8 for the *base* argument. In the absence
 34222 of a size modifier, the application shall ensure that the corresponding argument is a
 34223 pointer to **unsigned**.

34224 u Matches an optionally signed decimal integer, whose format is the same as expected for
 34225 the subject sequence of *wcstoul()* with the value 10 for the *base* argument. In the absence
 34226 of a size modifier, the application shall ensure that the corresponding argument is a
 34227 pointer to **unsigned**.

34228	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument.
34229		
34230		In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
34231		
34232	a, e, f, g	
34233		Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to float .
34234		
34235		
34236		
34237		If the <i>fwprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
34238		
34239		
34240	s	Matches a sequence of non-white-space wide characters. If no <code>l</code> (ell) qualifier is present, characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
34241		
34242		
34243		
34244		
34245		
34246	CX	
34247		
34248		
34249		If the <code>l</code> (ell) qualifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character, which shall be added automatically. If the <code>l</code> (ell) qualifier is present and the 'm' assignment-allocation character is present, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
34250		
34251		
34252	CX	
34253		
34254		
34255	[Matches a non-empty sequence of wide characters from a set of expected wide characters (the <i>scanset</i>). If no <code>l</code> (ell) qualifier is present, wide characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
34256		
34257		
34258		
34259		
34260		
34261		
34262	CX	
34263		
34264		If an <code>l</code> (ell) qualifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of wchar_t large enough to accept the sequence and the terminating null wide character. If an <code>l</code> (ell) qualifier is present and the 'm' assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
34265		
34266		
34267	CX	
34268		
34269		
34270		The conversion specification includes all subsequent wide characters in the <i>format</i> string up to and including the matching <right-square-bracket> (' <code>]</code> '). The wide characters between the square brackets (the <i>scanlist</i>) comprise the scanset, unless the wide character after the <left-square-bracket> is a <circumflex> (' <code>^</code> '), in which case the scanset contains all wide characters that do not appear in the scanlist between the <circumflex> and the <right-square-bracket>. If the conversion specification begins
34271		
34272		
34273		
34274		
34275		

34276			with "[]" or "[^]", the <right-square-bracket> is included in the scanlist and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the first <right-square-bracket> is the one that ends the conversion specification. If a '-' is in the scanlist and is not the first wide character, nor the second where the first wide character is a '^', nor the last wide character, the behavior is implementation-defined.
34277			
34278			
34279			
34280			
34281			
34282	c		Matches a sequence of wide characters of exactly the number specified by the field width (1 if no field width is present in the conversion specification).
34283			
34284			If no l (ell) length modifier is present, characters from the input field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object initialized to zero before the first wide character is converted. No null character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial element of a character array large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a char .
34285			
34286			
34287			
34288			
34289	CX		
34290			
34291			
34292			No null wide character is added. If an l (ell) length modifier is present and the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument shall be a pointer to the initial element of an array of wchar_t large enough to accept the sequence. If an l (ell) qualifier is present and the 'm' assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a wchar_t .
34293			
34294			
34295	CX		
34296			
34297			
34298	p		Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding <i>fwprintf()</i> functions. The application shall ensure that the corresponding argument is a pointer to a pointer to void . The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion is undefined.
34299			
34300			
34301			
34302			
34303			
34304			
34305	n		No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.
34306			
34307			
34308			
34309			
34310			
34311			
34312	XSI	C	Equivalent to lc.
34313	XSI	S	Equivalent to ls.
34314		%	Matches a single '%' wide character; no conversion or assignment shall occur. The complete conversion specification shall be %%.
34315			
34316			If a conversion specification is invalid, the behavior is undefined.
34317			The conversion specifiers A, E, F, G, and X are also valid and shall be equivalent to, respectively, a, e, f, g, and x.
34318			
34319			If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for %n) have been read (other than leading white-space, where permitted), execution of the current conversion
34320			
34321			

34322 specification shall terminate with an input failure. Otherwise, unless execution of the current
 34323 conversion specification is terminated with a matching failure, execution of the following
 34324 conversion specification (if any) shall be terminated with an input failure.

34325 Reaching the end of the string in *swscanf()* shall be equivalent to encountering end-of-file for
 34326 *fwscanf()*.

34327 If conversion terminates on a conflicting input, the offending input shall be left unread in the
 34328 input. Any trailing white space (including <newline>) shall be left unread unless matched by a
 34329 conversion specification. The success of literal matches and suppressed assignments is only
 34330 directly determinable via the %n conversion specification.

34331 CX The *fwscanf()* and *wscanf()* functions may mark the last data access timestamp of the file
 34332 associated with *stream* for update. The last data access timestamp shall be marked for update by
 34333 the first successful execution of *fgetc()*, *fgetws()*, *fwscanf()*, *getc()*, *getwchar()*, *vfwscanf()*,
 34334 *vwscanf()*, or *wscanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

34335 RETURN VALUE

34336 Upon successful completion, these functions shall return the number of successfully matched
 34337 and assigned input items; this number can be zero in the event of an early matching failure. If
 34338 the input ends before the first conversion (if any) has completed, and without a matching failure
 34339 having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has
 34340 CX completed, and without a matching failure having occurred, EOF shall be returned and *errno*
 34341 shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be
 34342 set.

34343 ERRORS

34344 For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetc()*.

34345 In addition, the *fwscanf()* function shall fail if:

34346 CX [EILSEQ] Input byte sequence does not form a valid character.

34347 [ENOMEM] Insufficient storage space is available.

34348 In addition, the *fwscanf()* function may fail if:

34349 CX [EINVAL] There are insufficient arguments.

34350 EXAMPLES

34351 The call:

```
34352 int i, n; float x; char name[50];
34353 n = wscanf(L"%d%f%s", &i, &x, name);
```

34354 with the input line:

```
34355 25 54.32E-1 Hamster
```

34356 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
 34357 "Hamster".

34358 The call:

```
34359 int i; float x; char name[50];
34360 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

34361 with input:

```
34362 56789 0123 56a72
```

34363 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to

34364 *getchar()* shall return the character 'a'.

34365 APPLICATION USAGE

34366 In format strings containing the '%' form of conversion specifications, each argument in the
34367 argument list is used exactly once.

34368 For functions that allocate memory as if by *malloc()*, the application should release such memory
34369 when it is no longer required by a call to *free()*. For *fwscanf()*, this is memory allocated via use of
34370 the 'm' assignment-allocation character.

34371 RATIONALE

34372 None.

34373 FUTURE DIRECTIONS

34374 None.

34375 SEE ALSO

34376 [Section 2.5](#) (on page 495), *getwc()*, *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*

34377 [XBD Chapter 7](#) (on page 135), [<inttypes.h>](#), [<stdio.h>](#), [<wchar.h>](#)

34378 CHANGE HISTORY

34379 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
34380 (E).

34381 Issue 6

34382 The normative text is updated to avoid use of the term “must” for application requirements.

34383 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

34384 The prototypes for *fwscanf()* and *swscanf()* are updated.

34385 The DESCRIPTION is updated.

34386 The hh, ll, j, t, and z length modifiers are added.

34387 The a, A, and F conversion characters are added.

34388 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
34389 specification” consistently.

34390 Issue 7

34391 Austin Group Interpretation 1003.1-2001 #170 is applied.

34392 SD5-XSH-ERN-132 is applied, adding the assignment-allocation character 'm'.

34393 Functionality relating to the "%n\$" form of conversion specification is moved from the XSI
34394 option to the Base.

34395 Changes are made related to support for finegrained timestamps.

34396 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
34397 *malloc()*.

34398 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0229 [302] and XSH/TC1-2008/0230
34399 [14] are applied.

34400 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0148 [73], XSH/TC2-2008/0149 [823],
34401 and XSH/TC2-2008/0150 [936] are applied.

34402 **NAME**

34403 gai_strerror — address and name information error description

34404 **SYNOPSIS**

34405 #include <netdb.h>

34406 const char *gai_strerror(int *ecode*);34407 **DESCRIPTION**34408 The *gai_strerror()* function shall return a text string describing an error value for the *getaddrinfo()*
34409 and *getnameinfo()* functions listed in the <netdb.h> header.34410 When the *ecode* argument is one of the following values listed in the <netdb.h> header:

34411	[EAI_AGAIN]	[EAI_NONAME]
34412	[EAI_BADFLAGS]	[EAI_OVERFLOW]
34413	[EAI_FAIL]	[EAI_SERVICE]
34414	[EAI_FAMILY]	[EAI_SOCKTYPE]
34415	[EAI_MEMORY]	[EAI_SYSTEM]

34416 the function return value shall point to a string describing the error. If the argument is not one
34417 of those values, the function shall return a pointer to a string whose contents indicate an
34418 unknown error.34419 **RETURN VALUE**34420 Upon successful completion, *gai_strerror()* shall return a pointer to an implementation-defined
34421 string.34422 **ERRORS**

34423 No errors are defined.

34424 **EXAMPLES**

34425 None.

34426 **APPLICATION USAGE**

34427 None.

34428 **RATIONALE**

34429 None.

34430 **FUTURE DIRECTIONS**

34431 None.

34432 **SEE ALSO**34433 [freeaddrinfo\(\)](#)

34434 XBD <netdb.h>

34435 **CHANGE HISTORY**

34436 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

34437 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from
34438 **char *** to **const char ***. This is for coordination with the IPnG Working Group.34439 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the
34440 [EAI_OVERFLOW] error code.

34441 **NAME**

34442 getaddrinfo — get address information

34443 **SYNOPSIS**

```
34444       #include <sys/socket.h>
34445       #include <netdb.h>
34446       int getaddrinfo(const char *restrict nodename,
34447                      const char *restrict servname,
34448                      const struct addrinfo *restrict hints,
34449                      struct addrinfo **restrict res);
```

34450 **DESCRIPTION**34451 Refer to *freeaddrinfo()*.

34452 **NAME**

34453 getc — get a byte from a stream

34454 **SYNOPSIS**

34455 #include <stdio.h>

34456 int getc(FILE *stream);

34457 **DESCRIPTION**

34458 CX The functionality described on this reference page is aligned with the ISO C standard. Any
34459 conflict between the requirements described here and the ISO C standard is unintentional. This
34460 volume of POSIX.1-2017 defers to the ISO C standard.

34461 The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it
34462 may evaluate *stream* more than once, so the argument should never be an expression with side-
34463 effects.

34464 **RETURN VALUE**34465 Refer to *fgetc()*.34466 **ERRORS**34467 Refer to *fgetc()*.34468 **EXAMPLES**

34469 None.

34470 **APPLICATION USAGE**

34471 If the integer value returned by *getc()* is stored into a variable of type **char** and then compared
34472 against the integer constant EOF, the comparison may never succeed, because sign-extension of
34473 a variable of type **char** on widening to integer is implementation-defined.

34474 Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with
34475 side-effects. In particular, *getc(*f++)* does not necessarily work as expected. Therefore, use of this
34476 function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.

34477 **RATIONALE**

34478 None.

34479 **FUTURE DIRECTIONS**

34480 None.

34481 **SEE ALSO**34482 Section 2.5 (on page 495), *fgetc()*

34483 XBD <stdio.h>

34484 **CHANGE HISTORY**

34485 First released in Issue 1. Derived from Issue 1 of the SVID.

34486 **Issue 7**

34487 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0231 [14] is applied.

34488 **NAME**

34489 getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked †'stdio with explicit client
34490 locking

34491 **SYNOPSIS**

```
34492 CX       #include <stdio.h>
34493       int getc_unlocked(FILE *stream);
34494       int getchar_unlocked(void);
34495       int putc_unlocked(int c, FILE *stream);
34496       int putchar_unlocked(int c);
```

34497 **DESCRIPTION**

34498 Versions of the functions *getc()*, *getchar()*, *putc()*, and *putchar()* respectively named
34499 *getc_unlocked()*, *getchar_unlocked()*, *putc_unlocked()*, and *putchar_unlocked()* shall be provided
34500 which are functionally equivalent to the original versions, with the exception that they are not
34501 required to be implemented in a fully thread-safe manner. They shall be thread-safe when used
34502 within a scope protected by *flockfile()* (or *ftrylockfile()*) and *funlockfile()*. These functions can
34503 safely be used in a multi-threaded program if and only if they are called while the invoking
34504 thread owns the (**FILE ***) object, as is the case after a successful call to the *flockfile()* or
34505 *ftrylockfile()* functions.

34506 If *getc_unlocked()* or *putc_unlocked()* are implemented as macros they may evaluate *stream* more
34507 than once, so the *stream* argument should never be an expression with side-effects.

34508 **RETURN VALUE**

34509 See *getc()*, *getchar()*, *putc()*, and *putchar()*.

34510 **ERRORS**

34511 See *getc()*, *getchar()*, *putc()*, and *putchar()*.

34512 **EXAMPLES**

34513 None.

34514 **APPLICATION USAGE**

34515 Since they may be implemented as macros, *getc_unlocked()* and *putc_unlocked()* may treat
34516 incorrectly a *stream* argument with side-effects. In particular, *getc_unlocked>(*f++)* and
34517 *putc_unlocked(c,*f++)* do not necessarily work as expected. Therefore, use of these functions in
34518 such situations should be preceded by the following statement as appropriate:

```
34519        #undef getc_unlocked
34520        #undef putc_unlocked
```

34521 **RATIONALE**

34522 Some I/O functions are typically implemented as macros for performance reasons (for example,
34523 *putc()* and *getc()*). For safety, they need to be synchronized, but it is often too expensive to
34524 synchronize on every character. Nevertheless, it was felt that the safety concerns were more
34525 important; consequently, the *getc()*, *getchar()*, *putc()*, and *putchar()* functions are required to be
34526 thread-safe. However, unlocked versions are also provided with names that clearly indicate the
34527 unsafe nature of their operation but can be used to exploit their higher performance. These
34528 unlocked versions can be safely used only within explicitly locked program regions, using
34529 exported locking primitives. In particular, a sequence such as:

```
34530        flockfile(fileptr);
34531        putc_unlocked('1', fileptr);
34532        putc_unlocked('\n', fileptr);
34533        fprintf(fileptr, "Line 2\n");
```

34534 `funlockfile(fileptr);`

34535 is permissible, and results in the text sequence:

34536 1

34537 Line 2

34538 being printed without being interspersed with output from other threads.

34539 It would be wrong to have the standard names such as `getc()`, `putc()`, and so on, map to the
34540 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still
34541 want to inspect all uses of `getc()`, `putc()`, and so on, by hand when converting existing code.
34542 Choosing the safe bindings as the default, at least, results in correct code and maintains the
34543 “atomicity at the function” invariant. To do otherwise would introduce gratuitous
34544 synchronization errors into converted code. Other routines that modify the `stdio (FILE *)`
34545 structures or buffers are also safely synchronized.

34546 Note that there is no need for functions of the form `getc_locked()`, `putc_locked()`, and so on, since
34547 this is the functionality of `getc()`, `putc()`, *et al.* It would be inappropriate to use a feature test
34548 macro to switch a macro definition of `getc()` between `getc_locked()` and `getc_unlocked()`, since the
34549 ISO C standard requires an actual function to exist, a function whose behavior could not be
34550 changed by the feature test macro. Also, providing both the `xxx_locked()` and `xxx_unlocked()`
34551 forms leads to the confusion of whether the suffix describes the behavior of the function or the
34552 circumstances under which it should be used.

34553 Three additional routines, `flockfile()`, `ftrylockfile()`, and `funlockfile()` (which may be macros), are
34554 provided to allow the user to delineate a sequence of I/O statements that are executed
34555 synchronously.

34556 The `ungetc()` function is infrequently called relative to the other functions/macros so no
34557 unlocked variation is needed.

34558 FUTURE DIRECTIONS

34559 None.

34560 SEE ALSO

34561 [Section 2.5](#) (on page 495), `flockfile()`, `getc()`, `getchar()`, `putc()`, `putchar()`

34562 XBD [<stdio.h>](#)

34563 CHANGE HISTORY

34564 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

34565 Issue 6

34566 These functions are marked as part of the Thread-Safe Functions option.

34567 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing
34568 how applications should be written to avoid the case when the functions are implemented as
34569 macros.

34570 Issue 7

34571 The `getc_unlocked()`, `getchar_unlocked()`, `putc_unlocked()`, and `putchar_unlocked()` functions are
34572 moved from the Thread-Safe Functions option to the Base.

34573 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0232 [395], XSH/TC1-2008/0233 [395],
34574 XSH/TC1-2008/0234 [395], and XSH/TC1-2008/0235 [14] are applied.

34575 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0151 [826] is applied.

34576 **NAME**34577 getchar — get a byte from a *stdin* stream34578 **SYNOPSIS**

34579 #include <stdio.h>

34580 int getchar(void);

34581 **DESCRIPTION**

34582 CX The functionality described on this reference page is aligned with the ISO C standard. Any
34583 conflict between the requirements described here and the ISO C standard is unintentional. This
34584 volume of POSIX.1-2017 defers to the ISO C standard.

34585 The *getchar()* function shall be equivalent to *getc(stdin)*.34586 **RETURN VALUE**34587 Refer to *fgetc()*.34588 **ERRORS**34589 Refer to *fgetc()*.34590 **EXAMPLES**

34591 None.

34592 **APPLICATION USAGE**

34593 If the integer value returned by *getchar()* is stored into a variable of type **char** and then
34594 compared against the integer constant EOF, the comparison may never succeed, because sign-
34595 extension of a variable of type **char** on widening to integer is implementation-defined.

34596 **RATIONALE**

34597 None.

34598 **FUTURE DIRECTIONS**

34599 None.

34600 **SEE ALSO**34601 [Section 2.5](#) (on page 495), *getc()*34602 XBD [<stdio.h>](#)34603 **CHANGE HISTORY**

34604 First released in Issue 1. Derived from Issue 1 of the SVID.

34605 **Issue 7**

34606 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0236 [14] is applied.

34607 **NAME**

34608 getchar_unlocked †'stdio with explicit client locking

34609 **SYNOPSIS**

34610 CX #include <stdio.h>

34611 int getchar_unlocked(void);

34612 **DESCRIPTION**

34613 Refer to *getc_unlocked()*.

34614 **NAME**

34615 getcwd — get the pathname of the current working directory

34616 **SYNOPSIS**

34617 #include <unistd.h>

34618 char *getcwd(char *buf, size_t size);

34619 **DESCRIPTION**

34620 The *getcwd()* function shall place an absolute pathname of the current working directory in the
 34621 array pointed to by *buf*, and return *buf*. The pathname shall contain no components that are dot
 34622 or dot-dot, or are symbolic links.

34623 If there are multiple pathnames that *getcwd()* could place in the array pointed to by *buf*, one
 34624 beginning with a single <slash> character and one or more beginning with two <slash>
 34625 characters, then *getcwd()* shall place the pathname beginning with a single <slash> character in
 34626 the array. The pathname shall not contain any unnecessary <slash> characters after the leading
 34627 one or two <slash> characters.

34628 The *size* argument is the size in bytes of the character array pointed to by the *buf* argument. If *buf*
 34629 is a null pointer, the behavior of *getcwd()* is unspecified.

34630 **RETURN VALUE**

34631 Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall
 34632 return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by
 34633 *buf* are then undefined.

34634 **ERRORS**34635 The *getcwd()* function shall fail if:34636 [EINVAL] The *size* argument is 0.34637 [ERANGE] The *size* argument is greater than 0, but is smaller than the length of the string
34638 +1.34639 The *getcwd()* function may fail if:34640 [EACCES] Search permission was denied for the current directory, or read or search
34641 permission was denied for a directory above the current directory in the file
34642 hierarchy.

34643 [ENOMEM] Insufficient storage space is available.

34644 **EXAMPLES**

34645 The following example uses {PATH_MAX} as the initial buffer size (unless it is indeterminate or
 34646 very large), and calls *getcwd()* with progressively larger buffers until it does not give an
 34647 [ERANGE] error.

34648 #include <stdlib.h>

34649 #include <errno.h>

34650 #include <unistd.h>

34651 ...

34652 long path_max;

34653 size_t size;

34654 char *buf;

34655 char *ptr;

34656 path_max = pathconf(".", _PC_PATH_MAX);

34657 if (path_max == -1)


```

34658         size = 1024;
34659     else if (path_max > 10240)
34660         size = 10240;
34661     else
34662         size = path_max;
34663     for (buf = ptr = NULL; ptr == NULL; size *= 2)
34664     {
34665         if ((buf = realloc(buf, size)) == NULL)
34666         {
34667             ... handle error ...
34668         }
34669         ptr = getcwd(buf, size);
34670         if (ptr == NULL && errno != ERANGE)
34671         {
34672             ... handle error ...
34673         }
34674     }
34675     ...
34676     free (buf);

```

34677 APPLICATION USAGE

34678 If the pathname obtained from *getcwd()* is longer than {PATH_MAX} bytes, it could produce an
34679 [ENAMETOOLONG] error if passed to *chdir()*. Therefore, in order to return to that directory it
34680 may be necessary to break the pathname into sections shorter than {PATH_MAX} bytes and call
34681 *chdir()* on each section in turn (the first section being an absolute pathname and subsequent
34682 sections being relative pathnames). A simpler way to handle saving and restoring the working
34683 directory when it may be deeper than {PATH_MAX} bytes in the file hierarchy is to use a file
34684 descriptor and *fchdir()*, rather than *getcwd()* and *chdir()*. However, the two methods do have
34685 some differences. The *fchdir()* approach causes the program to restore a working directory even
34686 if it has been renamed in the meantime, whereas the *chdir()* approach restores to a directory with
34687 the same name as the original, even if the directories were renamed in the meantime. Since the
34688 *fchdir()* approach does not access parent directories, it can succeed when *getcwd()* would fail
34689 due to permissions problems. In applications conforming to earlier versions of this standard, it
34690 was not possible to use the *fchdir()* approach when the working directory is searchable but not
34691 readable, as the only way to open a directory was with O_RDONLY, whereas the *getcwd()*
34692 approach can succeed in this case.

34693 RATIONALE

34694 Having *getcwd()* take no arguments and instead use the *malloc()* function to produce space for
34695 the returned argument was considered. The advantage is that *getcwd()* knows how big the
34696 working directory pathname is and can allocate an appropriate amount of space. But the
34697 programmer would have to use the *free()* function to free the resulting object, or each use of
34698 *getcwd()* would further reduce the available memory. Finally, *getcwd()* is taken from the SVID
34699 where it has the two arguments used in this volume of POSIX.1-2017.

34700 The older function *getwd()* was rejected for use in this context because it had only a buffer
34701 argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except
34702 to depend on the programmer to provide a large enough buffer.

34703 On some implementations, if *buf* is a null pointer, *getcwd()* may obtain *size* bytes of memory
34704 using *malloc()*. In this case, the pointer returned by *getcwd()* may be used as the argument in a
34705 subsequent call to *free()*. Invoking *getcwd()* with *buf* as a null pointer is not recommended in
34706 conforming applications.

34707 Earlier implementations of `getcwd()` sometimes generated pathnames like
34708 `"../../../../subdirname"` internally, using them to explore the path of ancestor directories
34709 back to the root. If one of these internal pathnames exceeded `{PATH_MAX}` in length, the
34710 implementation could fail with *errno* set to `[ENAMETOOLONG]`. This is no longer allowed.

34711 If a program is operating in a directory where some (grand)parent directory does not permit
34712 reading, `getcwd()` may fail, as in most implementations it must read the directory to determine
34713 the name of the file. This can occur if search, but not read, permission is granted in an
34714 intermediate directory, or if the program is placed in that directory by some more privileged
34715 process (for example, login). Including the `[EACCES]` error condition makes the reporting of the
34716 error consistent and warns the application developer that `getcwd()` can fail for reasons beyond
34717 the control of the application developer or user. Some implementations can avoid this
34718 occurrence (for example, by implementing `getcwd()` using `pwd`, where `pwd` is a set-user-root
34719 process), thus the error was made optional. Since this volume of POSIX.1-2017 permits the
34720 addition of other errors, this would be a common addition and yet one that applications could
34721 not be expected to deal with without this addition.

34722 **FUTURE DIRECTIONS**

34723 None.

34724 **SEE ALSO**

34725 [*malloc\(\)*](#)

34726 XBD [`<unistd.h>`](#)

34727 **CHANGE HISTORY**

34728 First released in Issue 1. Derived from Issue 1 of the SVID.

34729 **Issue 6**

34730 The following new requirements on POSIX implementations derive from alignment with the
34731 Single UNIX Specification:

34732 The `[ENOMEM]` optional error condition is added.

34733 **Issue 7**

34734 Austin Group Interpretation 1003.1-2001 #140 is applied, changing the text for consistency with
34735 the `pwd` utility, adding text to address the case where the current directory is deeper in the file
34736 hierarchy than `{PATH_MAX}` bytes, and adding the requirements relating to pathnames
34737 beginning with two `<slash>` characters.

34738 **NAME**

34739 getdate ‡convert user format date and time

34740 **SYNOPSIS**

```
34741 XSI #include <time.h>
34742 struct tm *getdate(const char *string);
```

34743 **DESCRIPTION**

34744 The *getdate()* function shall convert a string representation of a date or time into a broken-down
 34745 time.

34746 The external variable or macro *getdate_err*, which has type **int**, is used by *getdate()* to return error
 34747 values. It is unspecified whether *getdate_err* is a macro or an identifier declared with external
 34748 linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order
 34749 to access an actual object, or a program defines an identifier with the name *getdate_err*, the
 34750 behavior is undefined.

34751 Templates are used to parse and interpret the input string. The templates are contained in a text
 34752 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to
 34753 indicate the full pathname of the file that contains the templates. The first line in the template
 34754 that matches the input specification is used for interpretation and conversion into the internal
 34755 time format.

34756 The following conversion specifications shall be supported:

34757	%%	Equivalent to %.
34758	%a	Abbreviated weekday name.
34759	%A	Full weekday name.
34760	%b	Abbreviated month name.
34761	%B	Full month name.
34762	%c	Locale's appropriate date and time representation.
34763	%C	Century number [00,99]; leading zeros are permitted but not required.
34764	%d	Day of month [01,31]; the leading 0 is optional.
34765	%D	Date as %m/%d/%y.
34766	%e	Equivalent to %d.
34767	%h	Abbreviated month name.
34768	%H	Hour [00,23].
34769	%I	Hour [01,12].
34770	%m	Month number [01,12].
34771	%M	Minute [00,59].
34772	%n	Equivalent to <newline>.
34773	%p	Locale's equivalent of either AM or PM.
34774	%r	The locale's appropriate representation of time in AM and PM notation. In the POSIX 34775 locale, this shall be equivalent to %I:%M:%S %p.

34776	%R	Time as %H:%M.
34777	%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap second data must come from some external source.
34778		
34779		
34780	%t	Equivalent to <tab>.
34781	%T	Time as %H:%M:%S.
34782	%w	Weekday number (Sunday = [0,6]).
34783	%x	Locale's appropriate date representation.
34784	%X	Locale's appropriate time representation.
34785	%y	Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.
34786		
34787		
34788	Note:	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
34789		
34790		
34791	%Y	Year as "ccyy" (for example, 2001).
34792	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result. The <i>getdate()</i> function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month).
34793		
34794		
34795		
34796		The match between the template and input specification performed by <i>getdate()</i> shall be case-insensitive.
34797		
34798		The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the <i>LC_TIME</i> category (see <i>setlocale()</i>).
34799		
34800		
34801		Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra white space in either the template file or in <i>string</i> shall be ignored.
34802		
34803		
34804		The results are undefined if the conversion specifications %c, %x, and %X include unsupported conversion specifications.
34805		
34806		The following rules apply for converting the input specification into the internal format:
34807		If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if <i>localtime()</i> had been called.
34808		
34809		
34810		If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.
34811		
34812		If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.
34813		
34814		
34815		If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.
34816		

34817 If no date is given, the hour chosen shall be the hour, starting with the current hour and
 34818 moving into the future, which first matches the named hour.

34819 If a conversion specification in the DATEMSK file does not correspond to one of the conversion
 34820 specifications above, the behavior is unspecified.

34821 The *getdate()* function need not be thread-safe.

34822 RETURN VALUE

34823 Upon successful completion, *getdate()* shall return a pointer to a **struct tm**. Otherwise, it shall
 34824 return a null pointer and set *getdate_err* to indicate the error.

34825 ERRORS

34826 The *getdate()* function shall fail in the following cases, setting *getdate_err* to the value shown in
 34827 the list below. Any changes to *errno* are unspecified.

- 34828 1. The *DATEMSK* environment variable is null or undefined.
- 34829 2. The template file cannot be opened for reading.
- 34830 3. Failed to get file status information.
- 34831 4. The template file is not a regular file.
- 34832 5. An I/O error is encountered while reading the template file.
- 34833 6. Memory allocation failed (not enough memory available).
- 34834 7. There is no line in the template that matches the input.
- 34835 8. Invalid input specification. For example, February 31; or a time is specified that cannot be
 34836 represented in a **time_t** (representing the time in seconds since the Epoch).

34837 EXAMPLES

- 34838 1. The following example shows the possible contents of a template:

```
34839 %m
34840 %A %B %d, %Y, %H:%M:%S
34841 %A
34842 %B
34843 %m/%d/%y %I %p
34844 %d,%m,%Y %H:%M
34845 at %A the %dst of %B in %Y
34846 run job at %I %p,%B %dnd
34847 %A den %d. %B %Y %H.%M Uhr
```

- 34848 2. The following are examples of valid input specifications for the template in Example 1:

```
34849 getdate("10/1/87 4 PM");
34850 getdate("Friday");
34851 getdate("Friday September 18, 1987, 10:30:30");
34852 getdate("24,9,1986 10:30");
34853 getdate("at monday the 1st of december in 1986");
34854 getdate("run job at 3 PM, december 2nd");
```

34855 If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name
 34856 and *oktober* as a month name, the following would be valid:

```
34857 getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

34858 3. The following example shows how local date and time specification can be defined in the
34859 template:

Invocation	Line in Template
34860 getdate("11/27/86")	%m/%d/%y
34861 getdate("27.11.86")	%d.%m.%y
34862 getdate("86-11-27")	%y-%m-%d
34863 getdate("Friday 12:00:00")	%A %H:%M:%S

34865 4. The following examples help to illustrate the above rules assuming that the current date
34866 is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default C or
34867 POSIX locale:

Input	Line in Template	Date
34868 Mon	%a	Mon Sep 22 12:19:47 EDT 1986
34869 Sun	%a	Sun Sep 28 12:19:47 EDT 1986
34870 Fri	%a	Fri Sep 26 12:19:47 EDT 1986
34871 September	%B	Mon Sep 1 12:19:47 EDT 1986
34872 January	%B	Thu Jan 1 12:19:47 EST 1987
34873 December	%B	Mon Dec 1 12:19:47 EST 1986
34874 Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
34875 Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
34876 Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
34877 Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
34878 Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
34879 Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
34880 10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
34881 13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

34883 APPLICATION USAGE

34884 Although historical versions of *getdate()* did not require that **<time.h>** declare the external
34885 variable *getdate_err*, this volume of POSIX.1-2017 does require it. The standard developers
34886 encourage applications to remove declarations of *getdate_err* and instead incorporate the
34887 declaration by including **<time.h>**.

34888 Applications should use %Y (4-digit years) in preference to %y (2-digit years).

34889 RATIONALE

34890 In standard locales, the conversion specifications %c, %x, and %X do not include unsupported
34891 conversion specifiers and so the text regarding results being undefined is not a problem in that
34892 case.

34893 FUTURE DIRECTIONS

34894 None.

34895 SEE ALSO

34896 *ctime()*, *localtime()*, *setlocale()*, *strftime()*, *times()*

34897 XBD **<time.h>**

34898 CHANGE HISTORY

34899 First released in Issue 4, Version 2.

34900 Issue 5

34901 Moved from X/OPEN UNIX extension to BASE.

34902 The last paragraph of the DESCRIPTION is added.

34903 The %C conversion specification is added, and the exact meaning of the %y conversion
34904 specification is clarified in the DESCRIPTION.

34905 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

34906 The %R conversion specification is changed to follow historical practice.

34907 Issue 6

34908 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
34909 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
34910 functions.

34911 The description of %S is updated so that the valid range is [00,60] rather than [00,61].

34912 The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors
34913 for consistency with other functions.

34914 Issue 7

34915 Austin Group Interpretation 1003.1-2001 #156 is applied.

34916 The description of the *getdate_err* value is expanded.

34917 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0152 [796] is applied.

34918 **NAME**34919 `getdelim, getline` — read a delimited record from *stream*34920 **SYNOPSIS**

```

34921 CX      #include <stdio.h>
34922         ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
34923             int delimiter, FILE *restrict stream);
34924         ssize_t getline(char **restrict lineptr, size_t *restrict n,
34925             FILE *restrict stream);

```

34926 **DESCRIPTION**

34927 The `getdelim()` function shall read from *stream* until it encounters a character matching the
 34928 *delimiter* character. The *delimiter* argument is an **int**, the value of which the application shall
 34929 ensure is a character representable as an **unsigned char** of equal value that terminates the read
 34930 process. If the *delimiter* argument has any other value, the behavior is undefined.

34931 The application shall ensure that **lineptr* is a valid argument that could be passed to the `free()`
 34932 function. If **n* is non-zero, the application shall ensure that **lineptr* either points to an object of
 34933 size at least **n* bytes, or is a null pointer.

34934 If **lineptr* is a null pointer or if the object pointed to by **lineptr* is of insufficient size, an object
 34935 shall be allocated as if by `malloc()` or the object shall be reallocated as if by `realloc()`, respectively,
 34936 such that the object is large enough to hold the characters to be written to it, including the
 34937 terminating NUL, and **n* shall be set to the new size. If the object was allocated, or if the
 34938 reallocation operation moved the object, **lineptr* shall be updated to point to the new object or
 34939 new location. The characters read, including any delimiter, shall be stored in the object, and a
 34940 terminating NUL added when the delimiter or end-of-file is encountered.

34941 The `getline()` function shall be equivalent to the `getdelim()` function with the *delimiter* character
 34942 equal to the <newline> character.

34943 The `getdelim()` and `getline()` functions may mark the last data access timestamp of the file
 34944 associated with *stream* for update. The last data access timestamp shall be marked for update by
 34945 the first successful execution of `fgetc()`, `fgets()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `getdelim()`,
 34946 `getline()`, `gets()`, or `scanf()` using *stream* that returns data not supplied by a prior call to `ungetc()`.

34947 **RETURN VALUE**

34948 Upon successful completion, the `getline()` and `getdelim()` functions shall return the number of
 34949 bytes written into the buffer, including the delimiter character if one was encountered before
 34950 EOF, but excluding the terminating NUL character. If the end-of-file indicator for the stream is
 34951 set, or if no characters were read and the stream is at end-of-file, the end-of-file indicator for the
 34952 stream shall be set and the function shall return `-1`. If an error occurs, the error indicator for the
 34953 stream shall be set, and the function shall return `-1` and set `errno` to indicate the error.

34954 **ERRORS**

34955 For the conditions under which the `getdelim()` and `getline()` functions shall fail and may fail, refer
 34956 to `fgetc()`.

34957 In addition, these functions shall fail if:

34958 [EINVAL] *lineptr* or *n* is a null pointer.

34959 [ENOMEM] Insufficient memory is available.

34960 These functions may fail if:

34961 [EOVERFLOW] The number of bytes to be written into the buffer, including the delimiter
34962 character (if encountered), would exceed {SSIZE_MAX}.

34963 EXAMPLES

```
34964 #include <stdio.h>
34965 #include <stdlib.h>
34966 int main(void)
34967 {
34968     FILE *fp;
34969     char *line = NULL;
34970     size_t len = 0;
34971     ssize_t read;
34972     fp = fopen("/etc/motd", "r");
34973     if (fp == NULL)
34974         exit(1);
34975     while ((read = getline(&line, &len, fp)) != -1) {
34976         printf("Retrieved line of length %zu :\n", read);
34977         printf("%s", line);
34978     }
34979     if (ferror(fp)) {
34980         /* handle error */
34981     }
34982     free(line);
34983     fclose(fp);
34984     return 0;
34985 }
```

34986 APPLICATION USAGE

34987 Setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start
34988 parsing a file.

34989 The *ferror()* or *feof()* functions should be used to distinguish between an error condition and an
34990 end-of-file condition.

34991 Although a NUL terminator is always supplied after the line, note that *strlen(*lineptr)* will be
34992 smaller than the return value if the line contains embedded NUL characters.

34993 RATIONALE

34994 These functions are widely used to solve the problem that the *fgets()* function has with long
34995 lines. The functions automatically enlarge the target buffers if needed. These are especially
34996 useful since they reduce code needed for applications.

34997 FUTURE DIRECTIONS

34998 None.

34999 SEE ALSO

35000 [Section 2.5](#) (on page 495), [fgets\(\)](#), [fgetc\(\)](#), [free\(\)](#), [malloc\(\)](#), [realloc\(\)](#)

35001 XBD [<stdio.h>](#)

35002 **CHANGE HISTORY**

35003 First released in Issue 7.

35004 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0237 [14] is applied.

35005 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0153 [569], XSH/TC2-2008/0154 [571],
35006 and XSH/TC2-2008/0155 [570] are applied.

35007 **NAME**

35008 getegid — get the effective group ID

35009 **SYNOPSIS**

35010 #include <unistd.h>

35011 gid_t getegid(void);

35012 **DESCRIPTION**35013 The *getegid()* function shall return the effective group ID of the calling process. The *getegid()*
35014 function shall not modify *errno*.35015 **RETURN VALUE**35016 The *getegid()* function shall always be successful and no return value is reserved to indicate an
35017 error.35018 **ERRORS**

35019 No errors are defined.

35020 **EXAMPLES**

35021 None.

35022 **APPLICATION USAGE**

35023 None.

35024 **RATIONALE**35025 In a conforming environment, *getegid()* will always succeed. It is possible for implementations to
35026 provide an extension where a process in a non-conforming environment will not be associated
35027 with a user or group ID. It is recommended that such implementations return (**gid_t**)-1 and set
35028 *errno* to indicate such an environment; doing so does not violate this standard, since such an
35029 environment is already an extension.35030 **FUTURE DIRECTIONS**

35031 None.

35032 **SEE ALSO**35033 *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*35034 XBD <[sys/types.h](#)>, <[unistd.h](#)>35035 **CHANGE HISTORY**

35036 First released in Issue 1. Derived from Issue 1 of the SVID.

35037 **Issue 6**35038 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.35039 The following new requirements on POSIX implementations derive from alignment with the
35040 Single UNIX Specification:35041 The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was
35042 required for conforming implementations of previous POSIX specifications, it was not
35043 required for UNIX applications.35044 **Issue 7**

35045 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0156 [511] is applied.

35046 **NAME**

35047 getenv — get value of an environment variable

35048 **SYNOPSIS**

35049 #include <stdlib.h>

35050 char *getenv(const char *name);

35051 **DESCRIPTION**

35052 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 35053 conflict between the requirements described here and the ISO C standard is unintentional. This
 35054 volume of POSIX.1-2017 defers to the ISO C standard.

35055 The *getenv()* function shall search the environment of the calling process (see XBD [Chapter 8](#), on
 35056 page 173) for the environment variable *name* if it exists and return a pointer to the value of the
 35057 environment variable. If the specified environment variable cannot be found, a null pointer shall
 35058 be returned. The application shall ensure that it does not modify the string pointed to by the
 35059 *getenv()* function.

35060 CX The returned string pointer might be invalidated or the string content might be overwritten by a
 35061 CX subsequent call to *getenv()*, *setenv()*, *unsetenv()*,
 35062 XSI or (if supported) *putenv()* but they shall not be affected by a call to any other function in this
 35063 volume of POSIX.1-2017.

35064 CX The returned string pointer might also be invalidated if the calling thread is terminated.

35065 CX The *getenv()* function need not be thread-safe.

35066 **RETURN VALUE**

35067 Upon successful completion, *getenv()* shall return a pointer to a string containing the *value* for
 35068 the specified *name*. If the specified *name* cannot be found in the environment of the calling
 35069 process, a null pointer shall be returned.

35070 **ERRORS**

35071 No errors are defined.

35072 **EXAMPLES**35073 **Getting the Value of an Environment Variable**35074 The following example gets the value of the *HOME* environment variable.

```
35075        #include <stdlib.h>
35076        ...
35077        const char *name = "HOME";
35078        char *value;
35079        value = getenv(name);
```

35080 **APPLICATION USAGE**

35081 None.

35082 **RATIONALE**

35083 The *clearenv()* function was considered but rejected. The *putenv()* function has now been
 35084 included for alignment with the Single UNIX Specification.

35085 The *getenv()* function is inherently not thread-safe because it returns a value pointing to static
 35086 data.

35087 Conforming applications are required not to directly modify the pointers to which *environ*
 35088 points, but to use only the *setenv()*, *unsetenv()*, and *putenv()* functions, or assignment to *environ*

35089 itself, to manipulate the process environment. This constraint allows the implementation to
35090 properly manage the memory it allocates. This enables the implementation to free any space it
35091 has allocated to strings (and perhaps the pointers to them) stored in *environ* when *unsetenv()* is
35092 called. A C runtime start-up procedure (that which invokes *main()* and perhaps initializes
35093 *environ*) can also initialize a flag indicating that none of the environment has yet been copied to
35094 allocated storage, or that the separate table has not yet been initialized. If the application
35095 switches to a complete new environment by assigning a new value to *environ*, this can be
35096 detected by *getenv()*, *setenv()*, *unsetenv()*, or *putenv()* and the implementation can at that point
35097 reinitialize based on the new environment. (This may include copying the environment strings
35098 into a new array and assigning *environ* to point to it.)

35099 In fact, for higher performance of *getenv()*, implementations that do not provide *putenv()* could
35100 also maintain a separate copy of the environment in a data structure that could be searched
35101 much more quickly (such as an indexed hash table, or a binary tree), and update both it and the
35102 linear list at *environ* when *setenv()* or *unsetenv()* is invoked. On implementations that do provide
35103 *putenv()*, such a copy might still be worthwhile but would need to allow for the fact that
35104 applications can directly modify the content of environment strings added with *putenv()*. For
35105 example, if an environment string found by searching the copy is one that was added using
35106 *putenv()*, the implementation would need to check that the string in *environ* still has the same
35107 name (and value, if the copy includes values), and whenever searching the copy produces no
35108 match the implementation would then need to search each environment string in *environ* that
35109 was added using *putenv()* in case any of them have changed their names and now match. Thus,
35110 each use of *putenv()* to add to the environment would reduce the speed advantage of having the
35111 copy.

35112 Performance of *getenv()* can be important for applications which have large numbers of
35113 environment variables. Typically, applications like this use the environment as a resource
35114 database of user-configurable parameters. The fact that these variables are in the user's shell
35115 environment usually means that any other program that uses environment variables (such as *ls*,
35116 which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is
35117 similarly slowed down by the linear search through the variables.

35118 An implementation that maintains separate data structures, or even one that manages the
35119 memory it consumes, is not currently required as it was thought it would reduce consensus
35120 among implementors who do not want to change their historical implementations.

35121 **FUTURE DIRECTIONS**

35122 A future version may add one or more functions to access and modify the environment in a
35123 thread-safe manner.

35124 **SEE ALSO**

35125 [*exec*](#), [*putenv\(\)*](#), [*setenv\(\)*](#), [*unsetenv\(\)*](#)

35126 XBD [Chapter 8](#) (on page 173), [`<stdlib.h>`](#)

35127 **CHANGE HISTORY**

35128 First released in Issue 1. Derived from Issue 1 of the SVID.

35129 **Issue 5**

35130 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
35131 VALUE section.

35132 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

35133 **Issue 6**

35134 The following changes were made to align with the IEEE P1003.1a draft standard:

35135 References added to the new *setenv()* and *unsetenv()* functions.

35136 The normative text is updated to avoid use of the term “must” for application requirements.

35137 **Issue 7**

35138 Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv()* may
35139 also cause the string to be overwritten.

35140 Austin Group Interpretation 1003.1-2001 #148 is applied, adding the FUTURE DIRECTIONS.

35141 Austin Group Interpretation 1003.1-2001 #156 is applied.

35142 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0238 [75,428], XSH/TC1-2008/0239
35143 [167], and XSH/TC1-2008/0240 [167] are applied.

35144 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0157 [656] is applied.

35145 **NAME**

35146 geteuid — get the effective user ID

35147 **SYNOPSIS**

35148 #include <unistd.h>

35149 uid_t geteuid(void);

35150 **DESCRIPTION**35151 The *geteuid()* function shall return the effective user ID of the calling process. The *geteuid()*
35152 function shall not modify *errno*.35153 **RETURN VALUE**35154 The *geteuid()* function shall always be successful and no return value is reserved to indicate an
35155 error.35156 **ERRORS**

35157 No errors are defined.

35158 **EXAMPLES**

35159 None.

35160 **APPLICATION USAGE**

35161 None.

35162 **RATIONALE**35163 In a conforming environment, *geteuid()* will always succeed. It is possible for implementations
35164 to provide an extension where a process in a non-conforming environment will not be associated
35165 with a user or group ID. It is recommended that such implementations return **(uid_t)-1** and set
35166 *errno* to indicate such an environment; doing so does not violate this standard, since such an
35167 environment is already an extension.35168 **FUTURE DIRECTIONS**

35169 None.

35170 **SEE ALSO**35171 *getegid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*35172 XBD [<sys/types.h>](#), [<unistd.h>](#)35173 **CHANGE HISTORY**

35174 First released in Issue 1. Derived from Issue 1 of the SVID.

35175 **Issue 6**35176 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.35177 The following new requirements on POSIX implementations derive from alignment with the
35178 Single UNIX Specification:35179 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
35180 required for conforming implementations of previous POSIX specifications, it was not
35181 required for UNIX applications.35182 **Issue 7**

35183 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0158 [511] is applied.

35184 **NAME**

35185 getgid — get the real group ID

35186 **SYNOPSIS**

35187 #include <unistd.h>

35188 gid_t getgid(void);

35189 **DESCRIPTION**35190 The *getgid()* function shall return the real group ID of the calling process. The *getgid()* function
35191 shall not modify *errno*.35192 **RETURN VALUE**35193 The *getgid()* function shall always be successful and no return value is reserved to indicate an
35194 error.35195 **ERRORS**

35196 No errors are defined.

35197 **EXAMPLES**

35198 None.

35199 **APPLICATION USAGE**

35200 None.

35201 **RATIONALE**35202 In a conforming environment, *getgid()* will always succeed. It is possible for implementations to
35203 provide an extension where a process in a non-conforming environment will not be associated
35204 with a user or group ID. It is recommended that such implementations return (**gid_t**)-1 and set
35205 *errno* to indicate such an environment; doing so does not violate this standard, since such an
35206 environment is already an extension.35207 **FUTURE DIRECTIONS**

35208 None.

35209 **SEE ALSO**35210 *getegid()*, *geteuid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*35211 XBD <**sys/types.h**>, <**unistd.h**>35212 **CHANGE HISTORY**

35213 First released in Issue 1. Derived from Issue 1 of the SVID.

35214 **Issue 6**35215 In the SYNOPSIS, the optional include of the <**sys/types.h**> header is removed.35216 The following new requirements on POSIX implementations derive from alignment with the
35217 Single UNIX Specification:35218 The requirement to include <**sys/types.h**> has been removed. Although <**sys/types.h**> was
35219 required for conforming implementations of previous POSIX specifications, it was not
35220 required for UNIX applications.35221 **Issue 7**

35222 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0159 [511] is applied.

35223 **NAME**

35224 getgrent — get the group database entry

35225 **SYNOPSIS**

```
35226 XSI       #include <grp.h>  
35227       struct group *getgrent(void);
```

35228 **DESCRIPTION**

35229 Refer to *endgrent()*.

35230 **NAME**

35231 getgrgid, getgrgid_r — get group database entry for a group ID

35232 **SYNOPSIS**

```
35233 #include <grp.h>
35234 struct group *getgrgid(gid_t gid);
35235 int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
35236               size_t bufsize, struct group **result);
```

35237 **DESCRIPTION**35238 The *getgrgid()* function shall search the group database for an entry with a matching *gid*.35239 The *getgrgid()* function need not be thread-safe.35240 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid()*.
35241 If *getgrgid()* returns a null pointer and *errno* is set to non-zero, an error occurred.

35242 The *getgrgid_r()* function shall update the **group** structure pointed to by *grp* and store a pointer
35243 to that structure at the location pointed to by *result*. The structure shall contain an entry from
35244 the group database with a matching *gid*. Storage referenced by the group structure is allocated
35245 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
35246 *sysconf(_SC_GETGR_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial value
35247 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to
35248 by *result* on error or if the requested entry is not found.

35249 **RETURN VALUE**

35250 Upon successful completion, *getgrgid()* shall return a pointer to a **struct group** with the structure
35251 defined in **<grp.h>** with a matching entry if one is found. The *getgrgid()* function shall return a
35252 null pointer if either the requested entry was not found, or an error occurred. If the requested
35253 entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

35254 The application shall not modify the structure to which the return value points, nor any storage
35255 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
35256 structure, might be invalidated or the structure or the storage areas might be overwritten by a
35257 subsequent call to *getgrent()*, *getgrgid()*, or *getgrnam()*. The returned pointer, and pointers
35258 within the structure, might also be invalidated if the calling thread is terminated.

35259 If successful, the *getgrgid_r()* function shall return zero; otherwise, an error number shall be
35260 returned to indicate the error.

35261 **ERRORS**35262 The *getgrgid()* and *getgrgid_r()* functions may fail if:

- 35263 [EIO] An I/O error has occurred.
- 35264 [EINTR] A signal was caught during *getgrgid()*.
- 35265 [EMFILE] All file descriptors available to the process are currently open.
- 35266 [ENFILE] The maximum allowable number of files is currently open in the system.

35267 The *getgrgid_r()* function may fail if:

- 35268 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
35269 referenced by the resulting **group** structure.

35270 **EXAMPLES**

35271 Note that *sysconf*(*_SC_GETGR_R_SIZE_MAX*) may return *-1* if there is no hard limit on the size
 35272 of the buffer needed to store all the groups returned. This example shows how an application
 35273 can allocate a buffer of sufficient size to work with *getgrid_r*().

```

35274 long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
35275 size_t len;
35276 if (initlen == -1)
35277     /* Default initial length. */
35278     len = 1024;
35279 else
35280     len = (size_t) initlen;
35281 struct group result;
35282 struct group *resultp;
35283 char *buffer = malloc(len);
35284 if (buffer == NULL)
35285     ...handle error...
35286 int e;
35287 while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
35288     {
35289     size_t newlen = 2 * len;
35290     if (newlen < len)
35291         ...handle error...
35292     len = newlen;
35293     char *newbuffer = realloc(buffer, len);
35294     if (newbuffer == NULL)
35295         ...handle error...
35296     buffer = newbuffer;
35297     }
35298 if (e != 0)
35299     ...handle error...
35300 free (buffer);

```

35301 **Finding an Entry in the Group Database**

35302 The following example uses *getgrgid*() to search the group database for a group ID that was
 35303 previously stored in a *stat* structure, then prints out the group name if it is found. If the group is
 35304 not found, the program prints the numeric value of the group for the entry.

```

35305 #include <sys/types.h>
35306 #include <grp.h>
35307 #include <stdio.h>
35308 ...
35309 struct stat statbuf;
35310 struct group *grp;
35311 ...
35312 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
35313     printf(" %-8.8s", grp->gr_name);
35314 else
35315     printf(" %-8d", statbuf.st_gid);
35316 ...

```

35317 **APPLICATION USAGE**

35318 The `getgrgid_r()` function is thread-safe and shall return values in a user-supplied buffer instead
 35319 of possibly using a static data area that may be overwritten by each call.

35320 Portable applications should take into account that it is usual for an implementation to return `-1`
 35321 from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

35322 **RATIONALE**

35323 None.

35324 **FUTURE DIRECTIONS**

35325 None.

35326 **SEE ALSO**

35327 [*endgrent\(\)*](#), [*getgrnam\(\)*](#), [*sysconf\(\)*](#)

35328 XBD [*<grp.h>*](#), [*<sys/types.h>*](#)

35329 **CHANGE HISTORY**

35330 First released in Issue 1. Derived from System V Release 2.0.

35331 **Issue 5**

35332 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 35333 VALUE section.

35334 The `getgrgid_r()` function is included for alignment with the POSIX Threads Extension.

35335 A note indicating that the `getgrgid()` function need not be reentrant is added to the
 35336 DESCRIPTION.

35337 **Issue 6**

35338 The `getgrgid_r()` function is marked as part of the Thread-Safe Functions option.

35339 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 35340 describing matching the `gid`.

35341 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

35342 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

35343 The following new requirements on POSIX implementations derive from alignment with the
 35344 Single UNIX Specification:

35345 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
 35346 required for conforming implementations of previous POSIX specifications, it was not
 35347 required for UNIX applications.

35348 In the RETURN VALUE section, the requirement to set `errno` on error is added.

35349 The `[EIO]`, `[EINTR]`, `[EMFILE]`, and `[ENFILE]` optional error conditions are added.

35350 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 35351 its avoidance of possibly using a static data area.

35352 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
 35353 buffer from `bufsize` characters to bytes.

35354 **Issue 7**

35355 Austin Group Interpretation 1003.1-2001 #156 is applied.

35356 SD5-XBD-ERN-4 is applied, changing the definition of the `[EMFILE]` error.

35357 SD5-XSH-ERN-166 is applied.

- 35358 The *getgrgid_r()* function is moved from the Thread-Safe Functions option to the Base.
- 35359 A minor addition is made to the EXAMPLES section, reminding the application developer to
35360 free memory allocated as if by *malloc()*.
- 35361 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0241 [75] is applied.
- 35362 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0160 [808], XSH/TC2-2008/0161 [808],
35363 XSH/TC2-2008/0162 [656], and XSH/TC2-2008/0163 [808] are applied.

35364 **NAME**

35365 getgrnam, getgrnam_r — search group database for a name

35366 **SYNOPSIS**

```
35367 #include <grp.h>
35368 struct group *getgrnam(const char *name);
35369 int getgrnam_r(const char *name, struct group *grp, char *buffer,
35370 size_t bufsize, struct group **result);
```

35371 **DESCRIPTION**35372 The *getgrnam()* function shall search the group database for an entry with a matching *name*.35373 The *getgrnam()* function need not be thread-safe.35374 Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam()*.
35375 If *getgrnam()* returns a null pointer and *errno* is set to non-zero, an error occurred.35376 The *getgrnam_r()* function shall update the **group** structure pointed to by *grp* and store a pointer
35377 to that structure at the location pointed to by *result*. The structure shall contain an entry from
35378 the group database with a matching *name*. Storage referenced by the **group** structure is allocated
35379 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
35380 *sysconf(_SC_GETGR_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial value
35381 suggested for the size of this buffer. A null pointer is returned at the location pointed to by *result*
35382 on error or if the requested entry is not found.35383 **RETURN VALUE**35384 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in
35385 **<grp.h>** with a matching entry if one is found. The *getgrnam()* function shall return a null
35386 pointer if either the requested entry was not found, or an error occurred. If the requested entry
35387 was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.35388 The application shall not modify the structure to which the return value points, nor any storage
35389 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
35390 structure, might be invalidated or the structure or the storage areas might be overwritten by a
35391 subsequent call to *getgrent()*, *getgrgid()*, or *getgrnam()*. The returned pointer, and pointers
35392 within the structure, might also be invalidated if the calling thread is terminated.35393 The *getgrnam_r()* function shall return zero on success or if the requested entry was not found
35394 and no error has occurred. If any error has occurred, an error number shall be returned to
35395 indicate the error.35396 **ERRORS**35397 The *getgrnam()* and *getgrnam_r()* functions may fail if:

- 35398 [EIO] An I/O error has occurred.
- 35399 [EINTR] A signal was caught during *getgrnam()*.
- 35400 [EMFILE] All file descriptors available to the process are currently open.
- 35401 [ENFILE] The maximum allowable number of files is currently open in the system.

35402 The *getgrnam_r()* function may fail if:

- 35403 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
35404 referenced by the resulting **group** structure.

35405 **EXAMPLES**

35406 Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size
 35407 of the buffer needed to store all the groups returned. This example shows how an application
 35408 can allocate a buffer of sufficient size to work with `getgrnam_r()`.

```

35409     long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
35410     size_t len;
35411     if (initlen == -1)
35412         /* Default initial length. */
35413         len = 1024;
35414     else
35415         len = (size_t) initlen;
35416     struct group result;
35417     struct group *resultp;
35418     char *buffer = malloc(len);
35419     if (buffer == NULL)
35420         ...handle error...
35421     int e;
35422     while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
35423           == ERANGE)
35424     {
35425         size_t newlen = 2 * len;
35426         if (newlen < len)
35427             ...handle error...
35428         len = newlen;
35429         char *newbuffer = realloc(buffer, len);
35430         if (newbuffer == NULL)
35431             ...handle error...
35432         buffer = newbuffer;
35433     }
35434     if (e != 0)
35435         ...handle error...
35436     free (buffer);

```

35437 **APPLICATION USAGE**

35438 The `getgrnam_r()` function is thread-safe and shall return values in a user-supplied buffer instead
 35439 of possibly using a static data area that may be overwritten by each call.

35440 Portable applications should take into account that it is usual for an implementation to return `-1`
 35441 from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

35442 **RATIONALE**

35443 None.

35444 **FUTURE DIRECTIONS**

35445 None.

35446 **SEE ALSO**

35447 [*endgrent\(\)*](#), [*getgrgid\(\)*](#), [*sysconf\(\)*](#)

35448 XBD [*<grp.h>*](#), [*<sys/types.h>*](#)

35449 **CHANGE HISTORY**

35450 First released in Issue 1. Derived from System V Release 2.0.

35451 **Issue 5**35452 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
35453 VALUE section.35454 The `getgrnam_r()` function is included for alignment with the POSIX Threads Extension.35455 A note indicating that the `getgrnam()` function need not be reentrant is added to the
35456 DESCRIPTION.35457 **Issue 6**35458 The `getgrnam_r()` function is marked as part of the Thread-Safe Functions option.

35459 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

35460 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.35461 The following new requirements on POSIX implementations derive from alignment with the
35462 Single UNIX Specification:35463 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
35464 required for conforming implementations of previous POSIX specifications, it was not
35465 required for UNIX applications.35466 In the RETURN VALUE section, the requirement to set `errno` on error is added.

35467 The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

35468 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
35469 its avoidance of possibly using a static data area.35470 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
35471 buffer from `bufsize` characters to bytes.35472 **Issue 7**

35473 Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section.

35474 Austin Group Interpretation 1003.1-2001 #156 is applied.

35475 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

35476 SD5-XSH-ERN-166 is applied.

35477 The `getgrnam_r()` function is moved from the Thread-Safe Functions option to the Base.35478 A minor addition is made to the EXAMPLES section, reminding the application developer to
35479 free memory allocated as if by `malloc()`.

35480 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0242 [75] is applied.

35481 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0164 [808], XSH/TC2-2008/0165 [808],
35482 XSH/TC2-2008/0166 [656], and XSH/TC2-2008/0167 [808] are applied.

35483 **NAME**

35484 getgroups — get supplementary group IDs

35485 **SYNOPSIS**

```
35486 #include <unistd.h>
35487 int getgroups(int gidsetsize, gid_t grouplist[]);
```

35488 **DESCRIPTION**

35489 The *getgroups()* function shall fill in the array *grouplist* with the current supplementary group
 35490 IDs of the calling process. It is implementation-defined whether *getgroups()* also returns the
 35491 effective group ID in the *grouplist* array.

35492 The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual
 35493 number of group IDs stored in the array shall be returned. The values of array entries with
 35494 indices greater than or equal to the value returned are undefined.

35495 If *gidsetsize* is 0, *getgroups()* shall return the number of group IDs that it would otherwise return
 35496 without modifying the array pointed to by *grouplist*.

35497 If the effective group ID of the process is returned with the supplementary group IDs, the value
 35498 returned shall always be greater than or equal to one and less than or equal to the value of
 35499 {NGROUPS_MAX}+1.

35500 **RETURN VALUE**

35501 Upon successful completion, the number of supplementary group IDs shall be returned. A
 35502 return value of -1 indicates failure and *errno* shall be set to indicate the error.

35503 **ERRORS**

35504 The *getgroups()* function shall fail if:

35505 [EINVAL] The *gidsetsize* argument is non-zero and less than the number of group IDs
 35506 that would have been returned.

35507 **EXAMPLES**35508 **Getting the Supplementary Group IDs of the Calling Process**

35509 The following example places the current supplementary group IDs of the calling process into
 35510 the *group* array.

```
35511 #include <sys/types.h>
35512 #include <unistd.h>
35513 ...
35514 gid_t *group;
35515 int nogroups;
35516 long ngroups_max;
35517 ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
35518 group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));
35519 ngroups = getgroups(ngroups_max, group);
```

35520 **APPLICATION USAGE**

35521 None.

35522 **RATIONALE**

35523 The related function *setgroups()* is a privileged operation and therefore is not covered by this
 35524 volume of POSIX.1-2017.

35525 As implied by the definition of supplementary groups, the effective group ID may appear in the

35526 array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but
35527 the application needs to call *getegid()* to be sure of getting all of the information. Various
35528 implementation variations and administrative sequences cause the set of groups appearing in
35529 the result of *getgroups()* to vary in order and as to whether the effective group ID is included,
35530 even when the set of groups is the same (in the mathematical sense of “set”). (The history of a
35531 process and its parents could affect the details of the result.)

35532 Application developers should note that {NGROUPS_MAX} is not necessarily a constant on all
35533 implementations.

35534 **FUTURE DIRECTIONS**

35535 None.

35536 **SEE ALSO**

35537 *getegid()*, *setgid()*

35538 XBD <[sys/types.h](#)>, <[unistd.h](#)>

35539 **CHANGE HISTORY**

35540 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

35541 **Issue 5**

35542 Normative text previously in the APPLICATION USAGE section is moved to the
35543 DESCRIPTION.

35544 **Issue 6**

35545 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.

35546 The following new requirements on POSIX implementations derive from alignment with the
35547 Single UNIX Specification:

35548 The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was
35549 required for conforming implementations of previous POSIX specifications, it was not
35550 required for UNIX applications.

35551 A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS
35552 requirement.

35553 The following changes were made to align with the IEEE P1003.1a draft standard:

35554 An explanation is added that the effective group ID may be included in the supplementary
35555 group list.

35556 **NAME**
35557 gethostent †network host database functions

35558 **SYNOPSIS**
35559 #include <netdb.h>
35560 struct hostent *gethostent(void);

35561 **DESCRIPTION**
35562 Refer to *endhostent()*.

35563 **NAME**

35564 gethostid — get an identifier for the current host

35565 **SYNOPSIS**

```
35566 XSI #include <unistd.h>  
35567 long gethostid(void);
```

35568 **DESCRIPTION**35569 The *gethostid()* function shall retrieve a 32-bit identifier for the current host.35570 **RETURN VALUE**35571 Upon successful completion, *gethostid()* shall return an identifier for the current host.35572 **ERRORS**

35573 No errors are defined.

35574 **EXAMPLES**

35575 None.

35576 **APPLICATION USAGE**

35577 This volume of POSIX.1-2017 does not define the domain in which the return value is unique.

35578 **RATIONALE**

35579 None.

35580 **FUTURE DIRECTIONS**

35581 None.

35582 **SEE ALSO**35583 *initstate()*

35584 XBD <unistd.h>

35585 **CHANGE HISTORY**

35586 First released in Issue 4, Version 2.

35587 **Issue 5**

35588 Moved from X/OPEN UNIX extension to BASE.

35589 **NAME**

35590 gethostname — get name of current host

35591 **SYNOPSIS**

35592 #include <unistd.h>

35593 int gethostname(char *name, size_t namelen);

35594 **DESCRIPTION**

35595 The *gethostname()* function shall return the standard host name for the current machine. The
35596 *namelen* argument shall specify the size of the array pointed to by the *name* argument. The
35597 returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold
35598 the host name, then the returned name shall be truncated and it is unspecified whether the
35599 returned name is null-terminated.

35600 Host names are limited to {HOST_NAME_MAX} bytes.

35601 **RETURN VALUE**

35602 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

35603 **ERRORS**

35604 No errors are defined.

35605 **EXAMPLES**

35606 None.

35607 **APPLICATION USAGE**

35608 None.

35609 **RATIONALE**

35610 None.

35611 **FUTURE DIRECTIONS**

35612 None.

35613 **SEE ALSO**35614 *gethostid()*, *uname()*

35615 XBD <unistd.h>

35616 **CHANGE HISTORY**

35617 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35618 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from
35619 *socklen_t* to *size_t*.

35620 **NAME**

35621 getitimer, setitimer ‡get and set value of interval timer

35622 **SYNOPSIS**

```

35623 OB XSI #include <sys/time.h>
35624 int getitimer(int which, struct itimerval *value);
35625 int setitimer(int which, const struct itimerval *restrict value,
35626 struct itimerval *restrict ovalue);

```

35627 **DESCRIPTION**

35628 The *getitimer()* function shall store the current value of the timer specified by *which* into the
 35629 structure pointed to by *value*. The *setitimer()* function shall set the timer specified by *which* to the
 35630 value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store the
 35631 previous value of the timer in the structure pointed to by *ovalue*.

35632 A timer value is defined by the **itimerval** structure, specified in **<sys/time.h>**. If *it_value* is non-
 35633 zero, it shall indicate the time to the next timer expiration. If *it_interval* is non-zero, it shall
 35634 specify a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 shall
 35635 disable a timer, regardless of the value of *it_interval*. Setting *it_interval* to 0 shall disable a timer
 35636 after its next expiration (assuming *it_value* is non-zero).

35637 Implementations may place limitations on the granularity of timer values. For each interval
 35638 timer, if the requested timer value requires a finer granularity than the implementation supports,
 35639 the actual timer value shall be rounded up to the next supported value.

35640 An XSI-conforming implementation provides each process with at least three interval timers,
 35641 which are indicated by the *which* argument:

35642 **ITIMER_PROF** Decrements both in process virtual time and when the system is running
 35643 on behalf of the process. It is designed to be used by interpreters in
 35644 statistically profiling the execution of interpreted programs. Each time the
 35645 **ITIMER_PROF** timer expires, the **SIGPROF** signal is delivered.

35646 **ITIMER_REAL** Decrements in real time. A **SIGALRM** signal is delivered when this timer
 35647 expires.

35648 **ITIMER_VIRTUAL** Decrements in process virtual time. It runs only when the process is
 35649 executing. A **SIGVTALRM** signal is delivered when it expires.

35650 The interaction between *setitimer()* and *alarm()* or *sleep()* is unspecified.

35651 **RETURN VALUE**

35652 Upon successful completion, *getitimer()* or *setitimer()* shall return 0; otherwise, -1 shall be
 35653 returned and *errno* set to indicate the error.

35654 **ERRORS**

35655 The *setitimer()* function shall fail if:

35656 **[EINVAL]** The *value* argument is not in canonical form. (In canonical form, the number of
 35657 microseconds is a non-negative integer less than 1 000 000 and the number of
 35658 seconds is a non-negative integer.)

35659 The *getitimer()* and *setitimer()* functions may fail if:

35660 **[EINVAL]** The *which* argument is not recognized.

35661 **EXAMPLES**

35662 None.

35663 **APPLICATION USAGE**35664 Applications should use the *timer_gettime()* and *timer_settime()* functions instead of the
35665 obsolescent *gettimer()* and *setitimer()* functions, respectively.35666 **RATIONALE**

35667 None.

35668 **FUTURE DIRECTIONS**35669 The *gettimer()* and *setitimer()* functions may be removed in a future version.35670 **SEE ALSO**35671 *alarm()*, *exec*, *sleep()*, *timer_getoverrun()*35672 XBD <[signal.h](#)>, <[sys/time.h](#)>35673 **CHANGE HISTORY**

35674 First released in Issue 4, Version 2.

35675 **Issue 5**

35676 Moved from X/OPEN UNIX extension to BASE.

35677 **Issue 6**35678 The **restrict** keyword is added to the *setitimer()* prototype for alignment with the
35679 ISO/IEC 9899:1999 standard.35680 **Issue 7**35681 The *gettimer()* and *setitimer()* functions are marked obsolescent.

35682 **NAME**35683 getline — read a delimited record from *stream*35684 **SYNOPSIS**

```
35685 CX        #include <stdio.h>
35686            ssize_t getline(char **restrict lineptr, size_t *restrict n,
35687            FILE *restrict stream);
```

35688 **DESCRIPTION**35689 Refer to [getdelim\(\)](#).

35690 **NAME**

35691 getlogin, getlogin_r ¶get login name

35692 **SYNOPSIS**

35693 #include <unistd.h>

35694 char *getlogin(void);

35695 int getlogin_r(char *name, size_t namesize);

35696 **DESCRIPTION**

35697 The *getlogin()* function shall return a pointer to a string containing the user name associated by
 35698 the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-
 35699 null pointer, then that pointer points to the name that the user logged in under, even if there are
 35700 several login names with the same user ID.

35701 The *getlogin()* function need not be thread-safe.

35702 The *getlogin_r()* function shall put the name associated by the login activity with the controlling
 35703 terminal of the current process in the character array pointed to by *name*. The array is *namesize*
 35704 characters long and should have space for the name and the terminating null character. The
 35705 maximum size of the login name is {LOGIN_NAME_MAX}.

35706 If *getlogin_r()* is successful, *name* points to the name the user used at login, even if there are
 35707 several login names with the same user ID.

35708 The *getlogin()* and *getlogin_r()* functions may make use of file descriptors 0, 1, and 2 to find the
 35709 controlling terminal of the current process, examining each in turn until the terminal is found. If
 35710 in this case none of these three file descriptors is open to the controlling terminal, these functions
 35711 may fail. The method used to find the terminal associated with a file descriptor may depend on
 35712 the file descriptor being open to the actual terminal device, not */dev/tty*.

35713 **RETURN VALUE**

35714 Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer
 35715 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to
 35716 indicate the error.

35717 The application shall not modify the string returned. The returned pointer might be invalidated
 35718 or the string content might be overwritten by a subsequent call to *getlogin()*. The returned
 35719 pointer and the string content might also be invalidated if the calling thread is terminated.

35720 If successful, the *getlogin_r()* function shall return zero; otherwise, an error number shall be
 35721 returned to indicate the error.

35722 **ERRORS**

35723 These functions may fail if:

35724 [EMFILE] All file descriptors available to the process are currently open.

35725 [ENFILE] The maximum allowable number of files is currently open in the system.

35726 [ENOTTY] None of the file descriptors 0, 1, or 2 is open to the controlling terminal of the
 35727 current process.

35728 [ENXIO] The calling process has no controlling terminal.

35729 The *getlogin_r()* function may fail if:

35730 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
 35731 including the terminating null character.

35732 **EXAMPLES**35733 **Getting the User Login Name**

35734 The following example calls the *getlogin()* function to obtain the name of the user associated
 35735 with the calling process, and passes this information to the *getpwnam()* function to get the
 35736 associated user database information.

```

35737 #include <unistd.h>
35738 #include <sys/types.h>
35739 #include <pwd.h>
35740 #include <stdio.h>
35741 ...
35742 char *lgn;
35743 struct passwd *pw;
35744 ...
35745 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
35746     fprintf(stderr, "Get of user information failed.\n"); exit(1);
35747 }

```

35748 **APPLICATION USAGE**

35749 Three names associated with the current process can be determined: *getpwuid(geteuid())* shall
 35750 return the name associated with the effective user ID of the process; *getlogin()* shall return the
 35751 name associated with the current login activity; and *getpwuid(getuid())* shall return the name
 35752 associated with the real user ID of the process.

35753 The *getlogin_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 35754 possibly using a static data area that may be overwritten by each call.

35755 **RATIONALE**

35756 The *getlogin()* function returns a pointer to the user's login name. The same user ID may be
 35757 shared by several login names. If it is desired to get the user database entry that is used during
 35758 login, the result of *getlogin()* should be used to provide the argument to the *getpwnam()*
 35759 function. (This might be used to determine the user's login shell, particularly where a single user
 35760 has multiple login shells with distinct login names, but the same user ID.)

35761 The information provided by the *cuserid()* function, which was originally defined in the
 35762 POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
35763 getpwuid(geteuid())
```

35764 while the information provided by historical implementations of *cuserid()* can be obtained by:

```
35765 getpwuid(getuid())
```

35766 The thread-safe version of this function places the user name in a user-supplied buffer and
 35767 returns a non-zero value if it fails. The non-thread-safe version may return the name in a static
 35768 data area that may be overwritten by each call.

35769 **FUTURE DIRECTIONS**

35770 None.

35771 **SEE ALSO**

35772 *getpwnam()*, *getpwuid()*, *geteuid()*, *getuid()*

35773 XBD [<limits.h>](#), [<unistd.h>](#)

35774 **CHANGE HISTORY**

35775 First released in Issue 1. Derived from System V Release 2.0.

35776 **Issue 5**

35777 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
35778 VALUE section.

35779 The *getlogin_r()* function is included for alignment with the POSIX Threads Extension.

35780 A note indicating that the *getlogin()* function need not be reentrant is added to the
35781 DESCRIPTION.

35782 **Issue 6**

35783 The *getlogin_r()* function is marked as part of the Thread-Safe Functions option.

35784 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

35785 The following new requirements on POSIX implementations derive from alignment with the
35786 Single UNIX Specification:

35787 In the RETURN VALUE section, the requirement to set *errno* on error is added.

35788 The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

35789 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
35790 its avoidance of possibly using a static data area.

35791 **Issue 7**

35792 Austin Group Interpretation 1003.1-2001 #156 is applied.

35793 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

35794 The *getlogin_r()* function is moved from the Thread-Safe Functions option to the Base.

35795 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0243 [172], XSH/TC1-2008/0244 [75],
35796 and XSH/TC1-2008/0245 [172] are applied.

35797 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0168 [656] is applied.

35798 **NAME**35799 getmsg, getpmsg — receive next message from a STREAMS file (**STREAMS**)35800 **SYNOPSIS**

```

35801 OB XSR #include <stropts.h>
35802 int getmsg(int fildev, struct strbuf *restrict ctlptr,
35803           struct strbuf *restrict dataptr, int *restrict flagsp);
35804 int getpmsg(int fildev, struct strbuf *restrict ctlptr,
35805            struct strbuf *restrict dataptr, int *restrict bandp,
35806            int *restrict flagsp);

```

35807 **DESCRIPTION**

35808 The `getmsg()` function shall retrieve the contents of a message located at the head of the
 35809 STREAM head read queue associated with a STREAMS file and place the contents into one or
 35810 more buffers. The message contains either a data part, a control part, or both. The data and
 35811 control parts of the message shall be placed into separate buffers, as described below. The
 35812 semantics of each part are defined by the originator of the message.

35813 The `getpmsg()` function shall be equivalent to `getmsg()`, except that it provides finer control over
 35814 the priority of the messages received. Except where noted, all requirements on `getmsg()` also
 35815 pertain to `getpmsg()`.

35816 The *fildev* argument specifies a file descriptor referencing a STREAMS-based file.

35817 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points
 35818 to a buffer in which the data or control information is to be placed, and the *maxlen* member
 35819 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall
 35820 contain the number of bytes of data or control information actually received. The *len* member
 35821 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or
 35822 control information is present in the message.

35823 When `getmsg()` is called, *flagsp* should point to an integer that indicates the type of message the
 35824 process is able to receive. This is described further below.

35825 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold
 35826 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the
 35827 control (or data) part of the message shall not be processed and shall be left on the STREAM
 35828 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the
 35829 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part
 35830 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0
 35831 and there are more than 0 bytes of control (or data) information, that information shall be left on
 35832 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the
 35833 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder
 35834 of the message shall be left on the STREAM head read queue and a non-zero return value shall
 35835 be provided.

35836 By default, `getmsg()` shall process the first available message on the STREAM head read queue.
 35837 However, a process may choose to retrieve only high-priority messages by setting the integer
 35838 pointed to by *flagsp* to RS_HIPRI. In this case, `getmsg()` shall only process the next message if it is
 35839 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall
 35840 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS_HIPRI if a
 35841 high-priority message was retrieved, or 0 otherwise.

35842 For `getpmsg()`, the flags are different. The *flagsp* argument points to a bitmask with the following
 35843 mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like `getmsg()`,

35844 *getpmsg()* shall process the first available message on the STREAM head read queue. A process
 35845 may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to
 35846 MSG_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process
 35847 the next message if it is a high-priority message. In a similar manner, a process may choose to
 35848 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to
 35849 MSG_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case,
 35850 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than,
 35851 the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the
 35852 first message off the queue, the integer pointed to by *flagsp* should be set to MSG_ANY and the
 35853 integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-
 35854 priority message, the integer pointed to by *flagsp* shall be set to MSG_HIPRI and the integer
 35855 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to
 35856 MSG_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

35857 If O_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type
 35858 specified by *flagsp* is available at the front of the STREAM head read queue. If O_NONBLOCK is
 35859 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and
 35860 *getpmsg()* shall fail and set *errno* to [EAGAIN].

35861 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()*
 35862 shall continue to operate normally, as described above, until the STREAM head read queue is
 35863 empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

35864 RETURN VALUE

35865 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value
 35866 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates
 35867 that more control information is waiting for retrieval. A return value of MOREDATA indicates
 35868 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL
 35869 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and
 35870 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher
 35871 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()*
 35872 shall retrieve that higher-priority message before retrieving the remainder of the previous
 35873 message.

35874 If the high priority control part of the message is consumed, the message shall be placed back on
 35875 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve
 35876 the remainder of the message. If, however, a priority message arrives or already exists on the
 35877 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority
 35878 message before retrieving the remainder of the message that was put back.

35879 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

35880 ERRORS

35881 The *getmsg()* and *getpmsg()* functions shall fail if:

- | | | |
|-------|-----------|--|
| 35882 | [EAGAIN] | The O_NONBLOCK flag is set and no messages are available. |
| 35883 | [EBADF] | The <i>fildev</i> argument is not a valid file descriptor open for reading. |
| 35884 | [EBADMSG] | The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a
35885 pending file descriptor is at the STREAM head. |
| 35886 | [EINTR] | A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> . |
| 35887 | [EINVAL] | An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer
35888 referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a
35889 multiplexer. |

35890 [ENOSTR] A STREAM is not associated with *fdes*.

35891 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an
 35892 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
 35893 *getmsg()* or *getpmsg()* but reflects the prior error.

35894 EXAMPLES

35895 Getting Any Message

35896 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call
 35897 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,
 35898 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,
 35899 respectively.

```
35900 #include <stropts.h>
35901 ...
35902 int fd;
35903 char ctrlbuf[128];
35904 char databuf[512];
35905 struct strbuf ctrl;
35906 struct strbuf data;
35907 int flags = 0;
35908 int ret;

35909 ctrl.buf = ctrlbuf;
35910 ctrl.maxlen = sizeof(ctrlbuf);

35911 data.buf = databuf;
35912 data.maxlen = sizeof(databuf);

35913 ret = getmsg (fd, &ctrl, &data, &flags);
```

35914 Getting the First Message off the Queue

35915 In the following example, the call to *getpmsg()* retrieves the first available message on the
 35916 associated STREAM-head read queue.

```
35917 #include <stropts.h>
35918 ...
35919 int fd;
35920 char ctrlbuf[128];
35921 char databuf[512];
35922 struct strbuf ctrl;
35923 struct strbuf data;
35924 int band = 0;
35925 int flags = MSG_ANY;
35926 int ret;

35927 ctrl.buf = ctrlbuf;
35928 ctrl.maxlen = sizeof(ctrlbuf);

35929 data.buf = databuf;
35930 data.maxlen = sizeof(databuf);

35931 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

35932 **APPLICATION USAGE**

35933 None.

35934 **RATIONALE**

35935 None.

35936 **FUTURE DIRECTIONS**35937 The *getmsg()* and *getpmsg()* functions may be removed in a future version.35938 **SEE ALSO**35939 [Section 2.6](#) (on page 500), [poll\(\)](#), [putmsg\(\)](#), [read\(\)](#), [write\(\)](#)35940 XBD [<stropts.h>](#)35941 **CHANGE HISTORY**

35942 First released in Issue 4, Version 2.

35943 **Issue 5**

35944 Moved from X/OPEN UNIX extension to BASE.

35945 A paragraph regarding “high-priority control parts of messages” is added to the RETURN
35946 VALUE section.35947 **Issue 6**

35948 This function is marked as part of the XSI STREAMS Option Group.

35949 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the
35950 ISO/IEC 9899:1999 standard.35951 **Issue 7**35952 The *getmsg()* and *getpmsg()* functions are marked obsolescent.

35953 **NAME**35954 `getnameinfo` ‡get name information35955 **SYNOPSIS**35956 `#include <sys/socket.h>`35957 `#include <netdb.h>`

```
35958 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
35959 char *restrict node, socklen_t nodelen, char *restrict service,
35960 socklen_t servicelen, int flags);
```

35961 **DESCRIPTION**

35962 The `getnameinfo()` function shall translate a socket address to a node name and service location,
 35963 all of which are defined as in [freeaddrinfo\(\)](#).

35964 The `sa` argument points to a socket address structure to be translated. The `salen` argument
 35965 contains the length of the address pointed to by `sa`.

35966 IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible
 35967 IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node
 35968 name for that IPv4 address.

35969 If the address is the IPv6 unspecified address (" : : "), a lookup shall not be performed and the
 35970 behavior shall be the same as when the node's name cannot be located.

35971 If the `node` argument is non-NULL and the `nodelen` argument is non-zero, then the `node` argument
 35972 points to a buffer able to contain up to `nodelen` bytes that receives the node name as a null-
 35973 terminated string. If the `node` argument is NULL or the `nodelen` argument is zero, the node name
 35974 shall not be returned. If the node's name cannot be located, the numeric form of the address
 35975 contained in the socket address structure pointed to by the `sa` argument is returned instead of its
 35976 name.

35977 If the `service` argument is non-NULL and the `servicelen` argument is non-zero, then the `service`
 35978 argument points to a buffer able to contain up to `servicelen` bytes that receives the service name
 35979 as a null-terminated string. If the `service` argument is NULL or the `servicelen` argument is zero,
 35980 the service name shall not be returned. If the service's name cannot be located, the numeric form
 35981 of the service address (for example, its port number) shall be returned instead of its name.

35982 The `flags` argument is a flag that changes the default actions of the function. By default the fully-
 35983 qualified domain name (FQDN) for the host shall be returned, but:

35984 If the flag bit `NI_NOFQDN` is set, only the node name portion of the FQDN shall be
 35985 returned for local hosts.

35986 If the flag bit `NI_NUMERICHOST` is set, the numeric form of the address contained in the
 35987 socket address structure pointed to by the `sa` argument shall be returned instead of its
 35988 name.

35989 If the flag bit `NI_NAMEREQD` is set, an error shall be returned if the host's name cannot
 35990 be located.

35991 If the flag bit `NI_NUMERICSERV` is set, the numeric form of the service address shall be
 35992 returned (for example, its port number) instead of its name.

35993 If the flag bit `NI_NUMERICSCOPE` is set, the numeric form of the scope identifier shall be
 35994 returned (for example, interface index) instead of its name. This flag shall be ignored if the
 35995 `sa` argument is not an IPv6 address.

35996 If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service
 35997 (SOCK_DGRAM). The default behavior shall assume that the service is a stream service
 35998 (SOCK_STREAM).

35999 **Notes:**

- 36000 1. The two NI_NUMERICxxx flags are required to support the `-n` flag that many
 36001 commands provide.
- 36002 2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for
 36003 example, [512,514]) that represent different services for UDP and TCP.

36004 The `getnameinfo()` function shall be thread-safe.

36005 RETURN VALUE

36006 A zero return value for `getnameinfo()` indicates successful completion; a non-zero return value
 36007 indicates failure. The possible values for the failures are listed in the ERRORS section.

36008 Upon successful completion, `getnameinfo()` shall return the *node* and *service* names, if requested,
 36009 in the buffers provided. The returned names are always null-terminated strings.

36010 ERRORS

36011 The `getnameinfo()` function shall fail and return the corresponding value if:

36012 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

36013 [EAI_BADFLAGS]

36014 The *flags* had an invalid value.

36015 [EAI_FAIL] A non-recoverable error occurred.

36016 [EAI_FAMILY] The address family was not recognized or the address length was invalid for
 36017 the specified family.

36018 [EAI_MEMORY] There was a memory allocation failure.

36019 [EAI_NONAME] The name does not resolve for the supplied parameters.

36020 NI_NAMEREQD is set and the host's name cannot be located, or both
 36021 *nodename* and *servname* were null.

36022 [EAI_OVERFLOW]

36023 An argument buffer overflowed. The buffer pointed to by the *node* argument
 36024 or the *service* argument was too small.

36025 [EAI_SYSTEM] A system error occurred. The error code can be found in *errno*.

36026 EXAMPLES

36027 None.

36028 APPLICATION USAGE

36029 If the returned values are to be used as part of any further name resolution (for example, passed
 36030 to `getaddrinfo()`), applications should provide buffers large enough to store any result possible on
 36031 the system.

36032 Given the IPv4-mapped IPv6 address "`::ffff:1.2.3.4`", the implementation performs a
 36033 lookup as if the socket address structure contains the IPv4 address "`1.2.3.4`".

36034 The IPv6 unspecified address ("`:::`") and the IPv6 loopback address ("`:::1`") are not
 36035 IPv4-compatible addresses.

36036 **RATIONALE**

36037 None.

36038 **FUTURE DIRECTIONS**

36039 None.

36040 **SEE ALSO**36041 *endservent()*, *freeaddrinfo()*, *gai_strerror()*, *inet_ntop()*, *socket()*36042 XBD <[netdb.h](#)>, <[sys/socket.h](#)>36043 **CHANGE HISTORY**

36044 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36045 The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the
36046 ISO/IEC 9899:1999 standard.36047 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in
36048 the SYNOPSIS and DESCRIPTION for alignment with IPv6.36049 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the
36050 [EAI_OVERFLOW] error to the ERRORS section.36051 **Issue 7**36052 SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified
36053 address.36054 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0246 [284] and XSH/TC1-2008/0247
36055 [285] are applied.

36056 **NAME**

36057 getnetbyaddr, getnetbyname, getnetent †'network database functions

36058 **SYNOPSIS**

36059 #include <netdb.h>

36060 struct netent *getnetbyaddr(uint32_t net, int type);

36061 struct netent *getnetbyname(const char *name);

36062 struct netent *getnetent(void);

36063 **DESCRIPTION**

36064 Refer to *endnetent()*.

36065 **NAME**36066 getopt, optarg, opterr, optind, optopt \ddagger command option parsing36067 **SYNOPSIS**

36068 #include <unistd.h>

36069 int getopt(int argc, char * const argv[], const char *optstring);

36070 extern char *optarg;

36071 extern int opterr, optind, optopt;

36072 **DESCRIPTION**36073 The *getopt()* function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,
36074 6, 7, 9, and 10 in XBD [Section 12.2](#) (on page 216).36075 The parameters *argc* and *argv* are the argument count and argument array as passed to *main()*
36076 (see *exec()*). The argument *optstring* is a string of recognized option characters; if a character is
36077 followed by a <colon>, the option takes an argument. All option characters allowed by Utility
36078 Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as
36079 an extension.36080 The variable *optind* is the index of the next element of the *argv[]* vector to be processed. It shall
36081 be initialized to 1 by the system, and *getopt()* shall update it when it finishes with each element
36082 of *argv[]*. If the application sets *optind* to zero before calling *getopt()*, the behavior is unspecified.
36083 When an element of *argv[]* contains multiple option characters, it is unspecified how *getopt()*
36084 determines which options have already been processed.36085 The *getopt()* function shall return the next option character (if one is found) from *argv* that
36086 matches a character in *optstring*, if there is one that matches. If the option takes an argument,
36087 *getopt()* shall set the variable *optarg* to point to the option-argument as follows:

- 36088 1. If the option was the last character in the string pointed to by an element of
- argv*
- , then
-
- 36089
- optarg*
- shall contain the next element of
- argv*
- , and
- optind*
- shall be incremented by 2. If the
-
- 36090 resulting value of
- optind*
- is greater than
- argc*
- , this indicates a missing option-argument,
-
- 36091 and
- getopt()*
- shall return an error indication.
-
- 36092 2. Otherwise,
- optarg*
- shall point to the string following the option character in that element
-
- 36093 of
- argv*
- , and
- optind*
- shall be incremented by 1.

36094 If, when *getopt()* is called:36095 *argv[optind]* is a null pointer
36096 **argv[optind]* is not the character -
36097 *argv[optind]* points to the string "--"36098 *getopt()* shall return -1 without changing *optind*. If:36099 *argv[optind]* points to the string "--"36100 *getopt()* shall return -1 after incrementing *optind*.36101 If *getopt()* encounters an option character that is not contained in *optstring*, it shall return the
36102 <question-mark> ('?') character. If it detects a missing option-argument, it shall return the
36103 <colon> character (':') if the first character of *optstring* was a <colon>, or a <question-mark>
36104 character ('?') otherwise. In either case, *getopt()* shall set the variable *optopt* to the option
36105 character that caused the error. If the application has not set the variable *opterr* to 0 and the first
36106 character of *optstring* is not a <colon>, *getopt()* shall also print a diagnostic message to *stderr*
36107 in the format specified for the *getopts* utility, unless the *stderr* stream has wide orientation, in which
36108 case the behavior is undefined.36109 The *getopt()* function need not be thread-safe.

36110 **RETURN VALUE**

36111 The `getopt()` function shall return the next option character specified on the command line.

36112 A <colon> (':') shall be returned if `getopt()` detects a missing argument and the first character
36113 of `optstring` was a <colon> (':').

36114 A <question-mark> ('?') shall be returned if `getopt()` encounters an option character not in
36115 `optstring` or detects a missing argument and the first character of `optstring` was not a <colon>
36116 (':').

36117 Otherwise, `getopt()` shall return `-1` when all command line options are parsed.

36118 **ERRORS**

36119 If the application has not set the variable `opterr` to 0, the first character of `optstring` is not a
36120 <colon>, and a write error occurs while `getopt()` is printing a diagnostic message to `stderr`, then
36121 the error indicator for `stderr` shall be set; but `getopt()` shall still succeed and the value of `errno`
36122 after `getopt()` is unspecified.

36123 **EXAMPLES**36124 **Parsing Command Line Options**

36125 The following code fragment shows how you might process the arguments for a utility that can
36126 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require
36127 arguments:

```

36128 #include <stdio.h>
36129 #include <stdlib.h>
36130 #include <unistd.h>
36131
36132 int
36133 main(int argc, char *argv[ ])
36134 {
36135     int c;
36136     int bflg = 0, aflg = 0, errflg = 0;
36137     char *ifile;
36138     char *ofile;
36139     . . .
36140     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
36141         switch(c) {
36142             case 'a':
36143                 if (bflg)
36144                     errflg++;
36145                 else
36146                     aflg++;
36147                 break;
36148             case 'b':
36149                 if (aflg)
36150                     errflg++;
36151                 else
36152                     bflg++;
36153                 break;
36154             case 'f':
36155                 ifile = optarg;
36156                 break;
36157             case 'o':

```

```

36157         ofile = optarg;
36158         break;
36159     case ':':          /* -f or -o without operand */
36160         fprintf(stderr,
36161             "Option -%c requires an operand\n", optopt);
36162         errflg++;
36163         break;
36164     case '?':
36165         fprintf(stderr,
36166             "Unrecognized option: '-%c'\n", optopt);
36167         errflg++;
36168     }
36169 }
36170 if (errflg) {
36171     fprintf(stderr, "usage: . . . ");
36172     exit(2);
36173 }
36174 for ( ; optind < argc; optind++) {
36175     if (access(argv[optind], R_OK)) {
36176         . . .
36177     }

```

36178 This code accepts any of the following as equivalent:

```

36179 cmd -ao arg path path
36180 cmd -a -o arg path path
36181 cmd -o arg -a path path
36182 cmd -a -o arg -- path path
36183 cmd -a -oarg path path
36184 cmd -aoarg path path

```

36185 Selecting Options from the Command Line

36186 The following example selects the type of database routines the user wants to use based on the
36187 *Options* argument.

```

36188 #include <unistd.h>
36189 #include <string.h>
36190 ...
36191 const char *Options = "hdbtl";
36192 ...
36193 int dbtype, c;
36194 char *st;
36195 ...
36196 dbtype = 0;
36197 while ((c = getopt(argc, argv, Options)) != -1) {
36198     if ((st = strchr(Options, c)) != NULL) {
36199         dbtype = st - Options;
36200         break;
36201     }
36202 }

```

36203 **APPLICATION USAGE**

36204 The `getopt()` function is only required to support option characters included in Utility Syntax
 36205 Guideline 3. Many historical implementations of `getopt()` support other characters as options.
 36206 This is an allowed extension, but applications that use extensions are not maximally portable.
 36207 Note that support for multi-byte option characters is only possible when such characters can be
 36208 represented as type `int`.

36209 Applications which use wide-character output functions with `stderr` should ensure that any calls
 36210 to `getopt()` do not write to `stderr`, either by setting `opterr` to 0 or by ensuring the first character of
 36211 `optstring` is always a `<colon>`.

36212 While `ferror(stderr)` may be used to detect failures to write a diagnostic to `stderr` when `getopt()`
 36213 returns `'?'`, the value of `errno` is unspecified in such a condition. Applications desiring more
 36214 control over handling write failures should set `opterr` to 0 and independently perform output to
 36215 `stderr`, rather than relying on `getopt()` to do the output.

36216 **RATIONALE**

36217 The `optopt` variable represents historical practice and allows the application to obtain the identity
 36218 of the invalid option.

36219 The description has been written to make it clear that `getopt()`, like the `getopts` utility, deals with
 36220 option-arguments whether separated from the option by `<blank>` characters or not. Note that
 36221 the requirements on `getopt()` and `getopts` are more stringent than the Utility Syntax Guidelines.

36222 The `getopt()` function shall return `-1`, rather than EOF, so that `<stdio.h>` is not required.

36223 The special significance of a `<colon>` as the first character of `optstring` makes `getopt()` consistent
 36224 with the `getopts` utility. It allows an application to make a distinction between a missing
 36225 argument and an incorrect option letter without having to examine the option letter. It is true
 36226 that a missing argument can only be detected in one case, but that is a case that has to be
 36227 considered.

36228 **FUTURE DIRECTIONS**

36229 None.

36230 **SEE ALSO**

36231 [exec](#)

36232 [XBD Section 12.2](#) (on page 216), `<unistd.h>`

36233 [XCU `getopts`](#)

36234 **CHANGE HISTORY**

36235 First released in Issue 1. Derived from Issue 1 of the SVID.

36236 **Issue 5**

36237 A note indicating that the `getopt()` function need not be reentrant is added to the DESCRIPTION.

36238 **Issue 6**

36239 IEEE PASC Interpretation 1003.2 #150 is applied.

36240 Austin Group Interpretation 1003.1-2001 #156 is applied.

36241 **Issue 7**

36242 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0248 [318], XSH/TC1-2008/0249 [460],
 36243 XSH/TC1-2008/0250 [189], XSH/TC1-2008/0251 [189], XSH/TC1-2008/0252 [189], and
 36244 XSH/TC1-2008/0253 [460] are applied.

36245

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0169 [608] is applied.

36246 **NAME**

36247 getpeername †'get the name of the peer socket

36248 **SYNOPSIS**

36249 #include <sys/socket.h>

36250 int getpeername(int *socket*, struct sockaddr *restrict *address*,
36251 socklen_t *restrict *address_len*);36252 **DESCRIPTION**36253 The *getpeername()* function shall retrieve the peer address of the specified socket, store this
36254 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this
36255 address in the object pointed to by the *address_len* argument.36256 The *address_len* argument points to a **socklen_t** object which on input specifies the length of the
36257 supplied **sockaddr** structure, and on output specifies the length of the stored address. If the
36258 actual length of the address is greater than the length of the supplied **sockaddr** structure, the
36259 stored address shall be truncated.36260 If the protocol permits connections by unbound clients, and the peer is not bound, then the
36261 value stored in the object pointed to by *address* is unspecified.36262 **RETURN VALUE**36263 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
36264 indicate the error.36265 **ERRORS**36266 The *getpeername()* function shall fail if:36267 [EBADF] The *socket* argument is not a valid file descriptor.

36268 [EINVAL] The socket has been shut down.

36269 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

36270 [ENOTSOCK] The *socket* argument does not refer to a socket.

36271 [EOPNOTSUPP] The operation is not supported for the socket protocol.

36272 The *getpeername()* function may fail if:

36273 [ENOBUFS] Insufficient resources were available in the system to complete the call.

36274 **EXAMPLES**

36275 None.

36276 **APPLICATION USAGE**

36277 None.

36278 **RATIONALE**

36279 None.

36280 **FUTURE DIRECTIONS**

36281 None.

36282 **SEE ALSO**36283 [accept\(\)](#), [bind\(\)](#), [getsockname\(\)](#), [socket\(\)](#)36284 XBD [<sys/socket.h>](#)

36285 **CHANGE HISTORY**

36286 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36287 The **restrict** keyword is added to the *getpeername()* prototype for alignment with the
36288 ISO/IEC 9899:1999 standard.

36289 **Issue 7**

36290 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0254 [464] is applied.

36291 **NAME**

36292 getpgid — get the process group ID for a process

36293 **SYNOPSIS**

36294 #include <unistd.h>

36295 pid_t getpgid(pid_t pid);

36296 **DESCRIPTION**36297 The *getpgid()* function shall return the process group ID of the process whose process ID is equal
36298 to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.36299 **RETURN VALUE**36300 Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return
36301 (**pid_t**)-1 and set *errno* to indicate the error.36302 **ERRORS**36303 The *getpgid()* function shall fail if:36304 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the
36305 calling process, and the implementation does not allow access to the process
36306 group ID of that process from the calling process.36307 [ESRCH] There is no process with a process ID equal to *pid*.36308 The *getpgid()* function may fail if:36309 [EINVAL] The value of the *pid* argument is invalid.36310 **EXAMPLES**

36311 None.

36312 **APPLICATION USAGE**

36313 None.

36314 **RATIONALE**

36315 None.

36316 **FUTURE DIRECTIONS**

36317 None.

36318 **SEE ALSO**36319 *exec*, *fork()*, *getpgrp()*, *getpid()*, *getsid()*, *setpgid()*, *setsid()*36320 XBD <**unistd.h**>36321 **CHANGE HISTORY**

36322 First released in Issue 4, Version 2.

36323 **Issue 5**

36324 Moved from X/OPEN UNIX extension to BASE.

36325 **Issue 7**36326 The *getpgid()* function is moved from the XSI option to the Base.

36327 NAME

36328 getpgrp — get the process group ID of the calling process

36329 SYNOPSIS

36330 #include <unistd.h>
36331 pid_t getpgrp(void);

36332 DESCRIPTION

36333 The *getpgrp()* function shall return the process group ID of the calling process.

36334 RETURN VALUE

36335 The *getpgrp()* function shall always be successful and no return value is reserved to indicate an error.
36336

36337 ERRORS

36338 No errors are defined.

36339 EXAMPLES

36340 None.

36341 APPLICATION USAGE

36342 None.

36343 RATIONALE

36344 4.3 BSD provides a *getpgrp()* function that returns the process group ID for a specified process.
36345 Although this function supports job control, all known job control shells always specify the
36346 calling process with this function. Thus, the simpler System V *getpgrp()* suffices, and the added
36347 complexity of the 4.3 BSD *getpgrp()* is provided by the XSI extension *getpgid()*.

36348 FUTURE DIRECTIONS

36349 None.

36350 SEE ALSO

36351 *exec*, *fork()*, *getpgid()*, *getpid()*, *getppid()*, *kill()*, *setpgid()*, *setsid()*

36352 XBD <[sys/types.h](#)>, <[unistd.h](#)>

36353 CHANGE HISTORY

36354 First released in Issue 1. Derived from Issue 1 of the SVID.

36355 Issue 6

36356 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.

36357 The following new requirements on POSIX implementations derive from alignment with the
36358 Single UNIX Specification:

36359 The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was
36360 required for conforming implementations of previous POSIX specifications, it was not
36361 required for UNIX applications.

36362 **NAME**

36363 getpid — get the process ID

36364 **SYNOPSIS**

36365 #include <unistd.h>

36366 pid_t getpid(void);

36367 **DESCRIPTION**36368 The *getpid()* function shall return the process ID of the calling process.36369 **RETURN VALUE**36370 The *getpid()* function shall always be successful and no return value is reserved to indicate an error.36372 **ERRORS**

36373 No errors are defined.

36374 **EXAMPLES**

36375 None.

36376 **APPLICATION USAGE**

36377 None.

36378 **RATIONALE**

36379 None.

36380 **FUTURE DIRECTIONS**

36381 None.

36382 **SEE ALSO**36383 *exec*, *fork()*, *getpgrp()*, *getppid()*, *kill()*, *mkdtemp()*, *setpgid()*, *setsid()*36384 XBD <[sys/types.h](#)>, <[unistd.h](#)>36385 **CHANGE HISTORY**

36386 First released in Issue 1. Derived from Issue 1 of the SVID.

36387 **Issue 6**36388 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.

36389 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

36391 The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

36394 **NAME**

36395 getpmsg — receive next message from a STREAMS file

36396 **SYNOPSIS**

```
36397 OB XSI #include <stropts.h>
36398         int getpmsg(int fildes, struct strbuf *restrict ctlptr,
36399                 struct strbuf *restrict dataptr, int *restrict bandp,
36400                 int *restrict flagsp);
```

36401 **DESCRIPTION**36402 Refer to [getmsg\(\)](#).

36403 **NAME**

36404 getppid — get the parent process ID

36405 **SYNOPSIS**

36406 #include <unistd.h>

36407 pid_t getppid(void);

36408 **DESCRIPTION**36409 The *getppid()* function shall return the parent process ID of the calling process.36410 **RETURN VALUE**36411 The *getppid()* function shall always be successful and no return value is reserved to indicate an
36412 error.36413 **ERRORS**

36414 No errors are defined.

36415 **EXAMPLES**

36416 None.

36417 **APPLICATION USAGE**

36418 None.

36419 **RATIONALE**

36420 None.

36421 **FUTURE DIRECTIONS**

36422 None.

36423 **SEE ALSO**36424 *exec*, *fork()*, *getpgid()*, *getpgrp()*, *getpid()*, *kill()*, *setpgid()*, *setsid()*36425 XBD <[sys/types.h](#)>, <[unistd.h](#)>36426 **CHANGE HISTORY**

36427 First released in Issue 1. Derived from Issue 1 of the SVID.

36428 **Issue 6**36429 In the SYNOPSIS, the optional include of the <[sys/types.h](#)> header is removed.36430 The following new requirements on POSIX implementations derive from alignment with the
36431 Single UNIX Specification:36432 The requirement to include <[sys/types.h](#)> has been removed. Although <[sys/types.h](#)> was
36433 required for conforming implementations of previous POSIX specifications, it was not
36434 required for UNIX applications.

36435 **NAME**

36436 getpriority, setpriority ‡get and set the nice value

36437 **SYNOPSIS**

```
36438 XSI #include <sys/resource.h>
36439 int getpriority(int which, id_t who);
36440 int setpriority(int which, id_t who, int value);
```

36441 **DESCRIPTION**

36442 The *getpriority()* function shall obtain the nice value of a process, process group, or user. The
 36443 *setpriority()* function shall set the nice value of a process, process group, or user to
 36444 *value*+{NZERO}.

36445 Target processes are specified by the values of the *which* and *who* arguments. The *which*
 36446 argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER,
 36447 indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an
 36448 effective user ID, respectively. A 0 value for the *who* argument specifies the current process,
 36449 process group, or user.

36450 The nice value set with *setpriority()* shall be applied to the process. If the process is multi-
 36451 threaded, the nice value shall affect all system scope threads in the process.

36452 If more than one process is specified, *getpriority()* shall return value {NZERO} less than the
 36453 lowest nice value pertaining to any of the specified processes, and *setpriority()* shall set the nice
 36454 values of all of the specified processes to *value*+{NZERO}.

36455 The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling.
 36456 While the range of valid nice values is [0,{NZERO}*2-1], implementations may enforce more
 36457 restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value,
 36458 *setpriority()* shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater
 36459 than the system's highest supported nice value, *setpriority()* shall set the nice value to the
 36460 highest supported value.

36461 Only a process with appropriate privileges can lower its nice value.

36462 PS|TPS Any processes or threads using SCHED_FIFO or SCHED_RR shall be unaffected by a call to
 36463 *setpriority()*. This is not considered an error. A process which subsequently reverts to
 36464 SCHED_OTHER need not have its priority affected by such a *setpriority()* call.

36465 The effect of changing the nice value may vary depending on the process-scheduling algorithm
 36466 in effect.

36467 Since *getpriority()* can return the value -1 upon successful completion, it is necessary to set *errno*
 36468 to 0 prior to a call to *getpriority()*. If *getpriority()* returns the value -1, then *errno* can be checked
 36469 to see if an error occurred or if the value is a legitimate nice value.

36470 **RETURN VALUE**

36471 Upon successful completion, *getpriority()* shall return an integer in the range -{NZERO} to
 36472 {NZERO}-1. Otherwise, -1 shall be returned and *errno* set to indicate the error.

36473 Upon successful completion, *setpriority()* shall return 0; otherwise, -1 shall be returned and *errno*
 36474 set to indicate the error.

36475 **ERRORS**36476 The *getpriority()* and *setpriority()* functions shall fail if:36477 [ESRCH] No process could be located using the *which* and *who* argument values
36478 specified.36479 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*
36480 argument is not a valid process ID, process group ID, or user ID.36481 In addition, *setpriority()* may fail if:36482 [EPERM] A process was located, but neither the real nor effective user ID of the
36483 executing process match the effective user ID of the process whose nice value
36484 is being changed.36485 [EACCES] A request was made to change the nice value to a lower numeric value and the
36486 current process does not have appropriate privileges.36487 **EXAMPLES**36488 **Using *getpriority()***36489 The following example returns the current scheduling priority for the process ID returned by the
36490 call to *getpid()*.

```

36491 #include <sys/resource.h>
36492 ...
36493 int which = PRIO_PROCESS;
36494 id_t pid;
36495 int ret;

36496 pid = getpid();
36497 ret = getpriority(which, pid);

```

36498 **Using *setpriority()***

36499 The following example sets the priority for the current process ID to -20.

```

36500 #include <sys/resource.h>
36501 ...
36502 int which = PRIO_PROCESS;
36503 id_t pid;
36504 int priority = -20;
36505 int ret;

36506 pid = getpid();
36507 ret = setpriority(which, pid, priority);

```

36508 **APPLICATION USAGE**36509 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value
36510 $-\{\text{NZERO}\}$). The nice value is in the range $[0, 2 * \{\text{NZERO}\} - 1]$, while the return value for
36511 *getpriority()* and the third parameter for *setpriority()* are in the range $[-\{\text{NZERO}\}, \{\text{NZERO}\} - 1]$.36512 **RATIONALE**

36513 None.

36514 **FUTURE DIRECTIONS**

36515 None.

36516 **SEE ALSO**36517 *nice()*, *sched_get_priority_max()*, *sched_setscheduler()*36518 XBD <[sys/resource.h](#)>36519 **CHANGE HISTORY**

36520 First released in Issue 4, Version 2.

36521 **Issue 5**

36522 Moved from X/OPEN UNIX extension to BASE.

36523 The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion
36524 with functionality in the POSIX Realtime Extension.

36525 **NAME**

36526 getprotobyname, getprotobynumber, getprotent — network protocol database functions

36527 **SYNOPSIS**

36528 #include <netdb.h>

36529 struct protoent *getprotobyname(const char *name);

36530 struct protoent *getprotobynumber(int proto);

36531 struct protoent *getprotoent(void);

36532 **DESCRIPTION**

36533 Refer to *endprotoent()*.

36534 **NAME**

36535 getpwent ‡'get user database entry

36536 **SYNOPSIS**

```
36537 XSI       #include <pwd.h>  
36538       struct passwd *getpwent(void);
```

36539 **DESCRIPTION**36540 Refer to *endpwent()*.

36541 **NAME**

36542 getpwnam, getpwnam_r — search user database for a name

36543 **SYNOPSIS**

```
36544 #include <pwd.h>
36545 struct passwd *getpwnam(const char *name);
36546 int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
36547               size_t bufsize, struct passwd **result);
```

36548 **DESCRIPTION**36549 The *getpwnam()* function shall search the user database for an entry with a matching *name*.36550 The *getpwnam()* function need not be thread-safe.36551 Applications wishing to check for error situations should set *errno* to 0 before calling
36552 *getpwnam()*. If *getpwnam()* returns a null pointer and *errno* is non-zero, an error occurred.

36553 The *getpwnam_r()* function shall update the **passwd** structure pointed to by *pwd* and store a
36554 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
36555 from the user database with a matching *name*. Storage referenced by the structure is allocated
36556 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
36557 *sysconf(_SC_GETPW_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial value
36558 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to
36559 by *result* on error or if the requested entry is not found.

36560 **RETURN VALUE**

36561 The *getpwnam()* function shall return a pointer to a **struct passwd** with the structure as defined
36562 in **<pwd.h>** with a matching entry if found. A null pointer shall be returned if the requested
36563 entry is not found, or an error occurs. If the requested entry was not found, *errno* shall not be
36564 changed. On error, *errno* shall be set to indicate the error.

36565 The application shall not modify the structure to which the return value points, nor any storage
36566 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
36567 structure, might be invalidated or the structure or the storage areas might be overwritten by a
36568 subsequent call to *getpwent()*, *getpwnam()*, or *getpwuid()*. The returned pointer, and pointers
36569 within the structure, might also be invalidated if the calling thread is terminated.

36570 The *getpwnam_r()* function shall return zero on success or if the requested entry was not found
36571 and no error has occurred. If an error has occurred, an error number shall be returned to indicate
36572 the error.

36573 **ERRORS**

36574 These functions may fail if:

- 36575 [EIO] An I/O error has occurred.
- 36576 [EINTR] A signal was caught during *getpwnam()*.
- 36577 [EMFILE] All file descriptors available to the process are currently open.
- 36578 [ENFILE] The maximum allowable number of files is currently open in the system.

36579 The *getpwnam_r()* function may fail if:

- 36580 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
36581 referenced by the resulting **passwd** structure.

36582 **EXAMPLES**

36583 Note that `sysconf(_SC_GETPW_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size
 36584 of the buffer needed to store all the groups returned. This example shows how an application
 36585 can allocate a buffer of sufficient size to work with `getpwnam_r()`.

```

36586 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
36587 size_t len;
36588 if (initlen == -1)
36589     /* Default initial length. */
36590     len = 1024;
36591 else
36592     len = (size_t) initlen;
36593 struct passwd result;
36594 struct passwd *resultp;
36595 char *buffer = malloc(len);
36596 if (buffer == NULL)
36597     ...handle error...
36598 int e;
36599 while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
36600        == ERANGE)
36601     {
36602         size_t newlen = 2 * len;
36603         if (newlen < len)
36604             ...handle error...
36605         len = newlen;
36606         char *newbuffer = realloc(buffer, len);
36607         if (newbuffer == NULL)
36608             ...handle error...
36609         buffer = newbuffer;
36610     }
36611 if (e != 0)
36612     ...handle error...
36613 free (buffer);

```

36614 **Getting an Entry for the Login Name**

36615 The following example uses the `getlogin()` function to return the name of the user who logged in;
 36616 this information is passed to the `getpwnam()` function to get the user database entry for that user.

```

36617 #include <sys/types.h>
36618 #include <pwd.h>
36619 #include <unistd.h>
36620 #include <stdio.h>
36621 #include <stdlib.h>
36622 ...
36623 char *lgn;
36624 struct passwd *pw;
36625 ...
36626 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
36627     fprintf(stderr, "Get of user information failed.\n"); exit(1);
36628 }
36629 ...

```

36630 **APPLICATION USAGE**

36631 Three names associated with the current process can be determined: *getpwuid(getuid())* returns
 36632 the name associated with the effective user ID of the process; *getlogin()* returns the name
 36633 associated with the current login activity; and *getpwuid(getuid())* returns the name associated
 36634 with the real user ID of the process.

36635 The *getpwnam_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 36636 possibly using a static data area that may be overwritten by each call.

36637 Portable applications should take into account that it is usual for an implementation to return -1
 36638 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

36639 **RATIONALE**

36640 None.

36641 **FUTURE DIRECTIONS**

36642 None.

36643 **SEE ALSO**

36644 *getpwuid()*, *sysconf()*

36645 XBD `<pwd.h>`, `<sys/types.h>`

36646 **CHANGE HISTORY**

36647 First released in Issue 1. Derived from System V Release 2.0.

36648 **Issue 5**

36649 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 36650 VALUE section.

36651 The *getpwnam_r()* function is included for alignment with the POSIX Threads Extension.

36652 A note indicating that the *getpwnam()* function need not be reentrant is added to the
 36653 DESCRIPTION.

36654 **Issue 6**

36655 The *getpwnam_r()* function is marked as part of the Thread-Safe Functions option.

36656 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 36657 describing matching the *name*.

36658 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

36659 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

36660 The following new requirements on POSIX implementations derive from alignment with the
 36661 Single UNIX Specification:

36662 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
 36663 required for conforming implementations of previous POSIX specifications, it was not
 36664 required for UNIX applications.

36665 In the RETURN VALUE section, the requirement to set *errno* on error is added.

36666 The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

36667 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 36668 its avoidance of possibly using a static data area.

36669 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
 36670 buffer from *bufsize* characters to bytes.

36671 **Issue 7**

- 36672 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 36673 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.
- 36674 SD5-XSH-ERN-166 is applied.
- 36675 The *getpwnam_r()* function is moved from the Thread-Safe Functions option to the Base.
- 36676 A minor addition is made to the EXAMPLES section, reminding the application developer to free memory allocated as if by *malloc()*.
- 36677
- 36678 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0255 [75,428] is applied.
- 36679 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0170 [808] and XSH/TC2-2008/0171
- 36680 [656] are applied.

36681 **NAME**

36682 getpwuid, getpwuid_r — search user database for a user ID

36683 **SYNOPSIS**

```
36684     #include <pwd.h>
36685
36685     struct passwd *getpwuid(uid_t uid);
36686     int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
36687                   size_t bufsize, struct passwd **result);
```

36688 **DESCRIPTION**36689 The *getpwuid()* function shall search the user database for an entry with a matching *uid*.36690 The *getpwuid()* function need not be thread-safe.36691 Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid()*.
36692 If *getpwuid()* returns a null pointer and *errno* is set to non-zero, an error occurred.36693 The *getpwuid_r()* function shall update the **passwd** structure pointed to by *pwd* and store a
36694 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
36695 from the user database with a matching *uid*. Storage referenced by the structure is allocated
36696 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
36697 *sysconf(_SC_GETPW_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial value
36698 suggested for the size of this buffer. A null pointer shall be returned at the location pointed to
36699 by *result* on error or if the requested entry is not found.36700 **RETURN VALUE**36701 The *getpwuid()* function shall return a pointer to a **struct passwd** with the structure as defined in
36702 <pwd.h> with a matching entry if found. A null pointer shall be returned if the requested entry
36703 is not found, or an error occurs. If the requested entry was not found, *errno* shall not be changed.
36704 On error, *errno* shall be set to indicate the error.36705 The application shall not modify the structure to which the return value points, nor any storage
36706 areas pointed to by pointers within the structure. The returned pointer, and pointers within the
36707 structure, might be invalidated or the structure or the storage areas might be overwritten by a
36708 subsequent call to *getpwent()*, *getpwnam()*, or *getpwuid()*. The returned pointer, and pointers
36709 within the structure, might also be invalidated if the calling thread is terminated.36710 If successful, the *getpwuid_r()* function shall return zero; otherwise, an error number shall be
36711 returned to indicate the error.36712 **ERRORS**

36713 These functions may fail if:

- | | | |
|-------|----------|--|
| 36714 | [EIO] | An I/O error has occurred. |
| 36715 | [EINTR] | A signal was caught during <i>getpwuid()</i> . |
| 36716 | [EMFILE] | All file descriptors available to the process are currently open. |
| 36717 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

36718 The *getpwuid_r()* function may fail if:

- | | | |
|-------|----------|--|
| 36719 | [ERANGE] | Insufficient storage was supplied via <i>buffer</i> and <i>bufsize</i> to contain the data to be
36720 referenced by the resulting passwd structure. |
|-------|----------|--|

36721 **EXAMPLES**

36722 Note that `sysconf(_SC_GETPW_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size
 36723 of the buffer needed to store all the groups returned. This example shows how an application
 36724 can allocate a buffer of sufficient size to work with `getpwuid_r()`.

```

36725 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
36726 size_t len;
36727 if (initlen == -1)
36728     /* Default initial length. */
36729     len = 1024;
36730 else
36731     len = (size_t) initlen;
36732 struct passwd result;
36733 struct passwd *resultp;
36734 char *buffer = malloc(len);
36735 if (buffer == NULL)
36736     ...handle error...
36737 int e;
36738 while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
36739     {
36740     size_t newlen = 2 * len;
36741     if (newlen < len)
36742         ...handle error...
36743     len = newlen;
36744     char *newbuffer = realloc(buffer, len);
36745     if (newbuffer == NULL)
36746         ...handle error...
36747     buffer = newbuffer;
36748     }
36749 if (e != 0)
36750     ...handle error...
36751 free (buffer);

```

36752 **Getting an Entry for the Root User**

36753 The following example gets the user database entry for the user with user ID 0 (root).

```

36754 #include <sys/types.h>
36755 #include <pwd.h>
36756 ...
36757 uid_t id = 0;
36758 struct passwd *pwd;
36759 pwd = getpwuid(id);

```

36760 **Finding the Name for the Effective User ID**

36761 The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to
 36762 store the structure pointer returned by the call to the *getpwuid()* function. The *geteuid()* function
 36763 shall return the effective user ID of the calling process; this is used as the search criteria for the
 36764 *getpwuid()* function. The call to *getpwuid()* shall return a pointer to the structure containing that
 36765 user ID value.

```
36766 #include <unistd.h>
36767 #include <sys/types.h>
36768 #include <pwd.h>
36769 ...
36770 struct passwd *pws;
36771 pws = getpwuid(geteuid());
```

36772 **Finding an Entry in the User Database**

36773 The following example uses *getpwuid()* to search the user database for a user ID that was
 36774 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not
 36775 found, the program prints the numeric value of the user ID for the entry.

```
36776 #include <sys/types.h>
36777 #include <pwd.h>
36778 #include <stdio.h>
36779 ...
36780 struct stat statbuf;
36781 struct passwd *pwd;
36782 ...
36783 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
36784     printf(" %-8.8s", pwd->pw_name);
36785 else
36786     printf(" %-8d", statbuf.st_uid);
```

36787 **APPLICATION USAGE**

36788 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns
 36789 the name associated with the effective user ID of the process; *getlogin()* returns the name
 36790 associated with the current login activity; and *getpwuid(getuid())* returns the name associated
 36791 with the real user ID of the process.

36792 The *getpwuid_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 36793 possibly using a static data area that may be overwritten by each call.

36794 Portable applications should take into account that it is usual for an implementation to return -1
 36795 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

36796 **RATIONALE**

36797 None.

36798 **FUTURE DIRECTIONS**

36799 None.

36800 **SEE ALSO**

36801 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, *sysconf()*

36802 XBD `<pwd.h>`, `<sys/types.h>`

CHANGE HISTORY

36803 First released in Issue 1. Derived from System V Release 2.0.

Issue 5

36806 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
36807 VALUE section.

36808 The *getpwuid_r()* function is included for alignment with the POSIX Threads Extension.

36809 A note indicating that the *getpwuid()* function need not be reentrant is added to the
36810 DESCRIPTION.

Issue 6

36812 The *getpwuid_r()* function is marked as part of the Thread-Safe Functions option.

36813 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
36814 describing matching the *uid*.

36815 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

36816 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

36817 The following new requirements on POSIX implementations derive from alignment with the
36818 Single UNIX Specification:

36819 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
36820 required for conforming implementations of previous POSIX specifications, it was not
36821 required for UNIX applications.

36822 In the RETURN VALUE section, the requirement to set *errno* on error is added.

36823 The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

36824 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
36825 its avoidance of possibly using a static data area.

36826 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
36827 buffer from *bufsize* characters to bytes.

Issue 7

36829 Austin Group Interpretation 1003.1-2001 #156 is applied.

36830 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

36831 SD5-XSH-ERN-166 is applied.

36832 The *getpwuid_r()* function is moved from the Thread-Safe Functions option to the Base.

36833 A minor addition is made to the EXAMPLES section, reminding the application developer to
36834 free memory allocated as if by *malloc()*.

36835 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0256 [75] is applied.

36836 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0172 [808] and XSH/TC2-2008/0173
36837 [656] are applied.

36838 **NAME**

36839 getrlimit, setrlimit — control maximum resource consumption

36840 **SYNOPSIS**

```
36841 XSI #include <sys/resource.h>
36842 int getrlimit(int resource, struct rlimit *rlp);
36843 int setrlimit(int resource, const struct rlimit *rlp);
```

36844 **DESCRIPTION**

36845 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption
36846 of a variety of resources.

36847 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as
36848 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur*
36849 member specifies the current or soft limit and the *rlim_max* member specifies the maximum or
36850 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the
36851 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or
36852 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both
36853 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints
36854 described above.

36855 The value RLIM_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than
36856 any other limit value. If a call to *getrlimit()* returns RLIM_INFINITY for a resource, it means the
36857 implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any
36858 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource
36859 limit.

36860 The following resources are defined:

36861	RLIMIT_CORE	This is the maximum size of a core file, in bytes, that may be created by a process. A limit of 0 shall prevent the creation of a core file. If this limit is exceeded, the writing of a core file shall terminate at this size.
36862		
36863		
36864	RLIMIT_CPU	This is the maximum amount of CPU time, in seconds, used by a process. If this limit is exceeded, SIGXCPU shall be generated for the process. If the process is catching or ignoring SIGXCPU, or all threads belonging to that process are blocking SIGXCPU, the behavior is unspecified.
36865		
36866		
36867		
36868	RLIMIT_DATA	This is the maximum size of a data segment of the process, in bytes. If this limit is exceeded, the <i>malloc()</i> function shall fail with <i>errno</i> set to [ENOMEM].
36869		
36870		
36871	RLIMIT_FSIZE	This is the maximum size of a file, in bytes, that may be created by a process. If a write or truncate operation would cause this limit to be exceeded, SIGXFSZ shall be generated for the thread. If the thread is blocking, or the process is catching or ignoring SIGXFSZ, continued attempts to increase the size of a file from end-of-file to beyond the limit shall fail with <i>errno</i> set to [EFBIG].
36872		
36873		
36874		
36875		
36876		
36877	RLIMIT_NOFILE	This is a number one greater than the maximum value that the system may assign to a newly-created descriptor. If this limit is exceeded, functions that allocate a file descriptor shall fail with <i>errno</i> set to [EMFILE]. This limit constrains the number of file descriptors that a process may allocate.
36878		
36879		
36880		
36881		

36882 RLIMIT_STACK This is the maximum size of the initial thread's stack, in bytes. The
 36883 implementation does not automatically grow the stack beyond this limit.
 36884 If this limit is exceeded, SIGSEGV shall be generated for the thread. If the
 36885 thread is blocking SIGSEGV, or the process is ignoring or catching
 36886 SIGSEGV and has not made arrangements to use an alternate stack, the
 36887 disposition of SIGSEGV shall be set to SIG_DFL before it is generated.

36888 RLIMIT_AS This is the maximum size of total available memory of the process, in
 36889 bytes. If this limit is exceeded, the *malloc()* and *mmap()* functions shall fail
 36890 with *errno* set to [ENOMEM]. In addition, the automatic stack growth
 36891 fails with the effects outlined above.

36892 When using the *getrlimit()* function, if a resource limit can be represented correctly in an object
 36893 of type **rlim_t**, then its representation is returned; otherwise, if the value of the resource limit is
 36894 equal to that of the corresponding saved hard limit, the value returned shall be
 36895 RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.

36896 When using the *setrlimit()* function, if the requested new limit is RLIM_INFINITY, the new limit
 36897 shall be "no limit"; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit
 36898 shall be the corresponding saved hard limit; otherwise, if the requested new limit is
 36899 RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the
 36900 new limit shall be the requested value. In addition, if the corresponding saved limit can be
 36901 represented correctly in an object of type **rlim_t** then it shall be overwritten with the new limit.

36902 The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless
 36903 a previous call to *getrlimit()* returned that value as the soft or hard limit for the corresponding
 36904 resource limit.

36905 The determination of whether a limit can be correctly represented in an object of type **rlim_t** is
 36906 implementation-defined. For example, some implementations permit a limit whose value is
 36907 greater than RLIM_INFINITY and others do not.

36908 The *exec* family of functions shall cause resource limits to be saved.

36909 RETURN VALUE

36910 Upon successful completion, *getrlimit()* and *setrlimit()* shall return 0. Otherwise, these functions
 36911 shall return -1 and set *errno* to indicate the error.

36912 ERRORS

36913 The *getrlimit()* and *setrlimit()* functions shall fail if:

36914 [EINVAL] An invalid *resource* was specified; or in a *setrlimit()* call, the new *rlim_cur*
 36915 exceeds the new *rlim_max*.

36916 [EPERM] The limit specified to *setrlimit()* would have raised the maximum limit value,
 36917 and the calling process does not have appropriate privileges.

36918 The *setrlimit()* function may fail if:

36919 [EINVAL] The limit specified cannot be lowered because current usage is already higher
 36920 than the limit.

36921 EXAMPLES

36922 None.

36923 APPLICATION USAGE

36924 If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value
36925 of {_POSIX_OPEN_MAX} from <limits.h>, unexpected behavior may occur.

36926 If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the
36927 highest currently open file descriptor +1, unexpected behavior may occur.

36928 RATIONALE

36929 It should be noted that RLIMIT_STACK applies “at least” to the stack of the initial thread in the
36930 process, and not to the sum of all the stacks in the process, as that would be very limiting unless
36931 the value is so big as to provide no value at all with a single thread.

36932 FUTURE DIRECTIONS

36933 None.

36934 SEE ALSO

36935 *exec, fork(), malloc(), open(), sigaltstack(), sysconf(), ulimit()*

36936 XBD <stropts.h>, <sys/resource.h>

36937 CHANGE HISTORY

36938 First released in Issue 4, Version 2.

36939 Issue 5

36940 Moved from X/OPEN UNIX extension to BASE.

36941 An APPLICATION USAGE section is added.

36942 Large File Summit extensions are added.

36943 Issue 6

36944 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for
36945 RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing
36946 with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of
36947 a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest
36948 currently open file descriptor +1.

36949 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/46 is applied, updating the definition of
36950 RLIMIT_STACK in the DESCRIPTION from “the maximum size of a process stack” to “the
36951 maximum size of the initial thread’s stack”. Text is added to the RATIONALE section.

36952 **NAME**

36953 getrusage — get information about resource utilization

36954 **SYNOPSIS**

```
36955 XSI      #include <sys/resource.h>
36956      int getrusage(int who, struct rusage *r_usage);
```

36957 **DESCRIPTION**

36958 The *getrusage()* function shall provide measures of the resources used by the current process or
 36959 its terminated and waited-for child processes. If the value of the *who* argument is
 36960 RUSAGE_SELF, information shall be returned about resources used by the current process. If the
 36961 value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about
 36962 resources used by the terminated and waited-for children of the current process. If the child is
 36963 never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to
 36964 SIG_IGN), the resource information for the child process is discarded and not included in the
 36965 resource information provided by *getrusage()*.

36966 The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned
 36967 information is stored.

36968 **RETURN VALUE**

36969 Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*
 36970 set to indicate the error.

36971 **ERRORS**

36972 The *getrusage()* function shall fail if:

36973 [EINVAL] The value of the *who* argument is not valid.

36974 **EXAMPLES**36975 **Using getrusage()**

36976 The following example returns information about the resources used by the current process.

```
36977 #include <sys/resource.h>
36978 ...
36979 int who = RUSAGE_SELF;
36980 struct rusage usage;
36981 int ret;
36982
36983 ret = getrusage(who, &usage);
```

36983 **APPLICATION USAGE**

36984 None.

36985 **RATIONALE**

36986 None.

36987 **FUTURE DIRECTIONS**

36988 None.

36989 **SEE ALSO**

36990 [exit\(\)](#), [sigaction\(\)](#), [time\(\)](#), [times\(\)](#), [wait\(\)](#)

36991 XBD [<sys/resource.h>](#)

36992 **CHANGE HISTORY**

36993 First released in Issue 4, Version 2.

36994 **Issue 5**

36995 Moved from X/OPEN UNIX extension to BASE.

36996 **NAME**36997 gets — get a string from a *stdin* stream36998 **SYNOPSIS**

```
36999 OB #include <stdio.h>
37000 char *gets(char *s);
```

37001 **DESCRIPTION**

37002 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37003 conflict between the requirements described here and the ISO C standard is unintentional. This
 37004 volume of POSIX.1-2017 defers to the ISO C standard.

37005 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed
 37006 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall
 37007 be discarded and a null byte shall be placed immediately after the last byte read into the array.

37008 CX The *gets()* function may mark the last data access timestamp of the file associated with *stream* for
 37009 update. The last data access timestamp shall be marked for update by the first successful
 37010 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *getline()*, *gets()*, or
 37011 *scanf()* using *stream* that returns data not supplied by a prior call to *ungetc()*.

37012 **RETURN VALUE**

37013 Upon successful completion, *gets()* shall return *s*. If the end-of-file indicator for the stream is set,
 37014 or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *gets()*
 37015 shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set,
 37016 CX *gets()* shall return a null pointer, and set *errno* to indicate the error.

37017 **ERRORS**37018 Refer to *fgetc()*.37019 **EXAMPLES**

37020 None.

37021 **APPLICATION USAGE**

37022 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of
 37023 *fgets()* is recommended.

37024 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is
 37025 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in
 37026 such a way as to cause applications to fail, or possible system security violations.

37027 Applications should use the *fgets()* function instead of the obsolescent *gets()* function.

37028 **RATIONALE**

37029 The standard developers decided to mark the *gets()* function as obsolescent even though it is in
 37030 the ISO C standard due to the possibility of buffer overflow.

37031 **FUTURE DIRECTIONS**37032 The *gets()* function may be removed in a future version.37033 **SEE ALSO**37034 [Section 2.5](#) (on page 495), *feof()*, *ferror()*, *fgets()*37035 XBD [<stdio.h>](#)

37036 CHANGE HISTORY

37037 First released in Issue 1. Derived from Issue 1 of the SVID.

37038 Issue 6

37039 Extensions beyond the ISO C standard are marked.

37040 Issue 7

37041 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

37042 The *gets()* function is marked obsolescent.

37043 Changes are made related to support for finegrained timestamps.

37044 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0257 [14] is applied.

37045 **NAME**

37046 getservbyname, getservbyport, getservent ǂnetwork services database functions

37047 **SYNOPSIS**

37048 #include <netdb.h>

37049 struct servent *getservbyname(const char *name, const char *proto);

37050 struct servent *getservbyport(int port, const char *proto);

37051 struct servent *getservent(void);

37052 **DESCRIPTION**37053 Refer to *endservent()*.

37054 **NAME**37055 *getsid* — get the process group ID of a session leader37056 **SYNOPSIS**

37057 #include <unistd.h>

37058 pid_t getsid(pid_t *pid*);37059 **DESCRIPTION**37060 The *getsid()* function shall obtain the process group ID of the process that is the session leader of
37061 the process specified by *pid*. If *pid* is (**pid_t**)0, it specifies the calling process.37062 **RETURN VALUE**37063 Upon successful completion, *getsid()* shall return the process group ID of the session leader of
37064 the specified process. Otherwise, it shall return `-1` and set *errno* to indicate the error.37065 **ERRORS**37066 The *getsid()* function shall fail if:37067 [**EPERM**] The process specified by *pid* is not in the same session as the calling process,
37068 and the implementation does not allow access to the process group ID of the
37069 session leader of that process from the calling process.37070 [**ESRCH**] There is no process with a process ID equal to *pid*.37071 **EXAMPLES**

37072 None.

37073 **APPLICATION USAGE**

37074 None.

37075 **RATIONALE**

37076 None.

37077 **FUTURE DIRECTIONS**

37078 None.

37079 **SEE ALSO**37080 *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*37081 XBD <**unistd.h**>37082 **CHANGE HISTORY**

37083 First released in Issue 4, Version 2.

37084 **Issue 5**

37085 Moved from X/OPEN UNIX extension to BASE.

37086 **Issue 7**37087 The *getsid()* function is moved from the XSI option to the Base.

37088 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0258 [421] is applied.

37089 **NAME**37090 `getsockname` ¶get the socket name37091 **SYNOPSIS**

```
37092 #include <sys/socket.h>
37093 int getsockname(int socket, struct sockaddr *restrict address,
37094                 socklen_t *restrict address_len);
```

37095 **DESCRIPTION**

37096 The `getsockname()` function shall retrieve the locally-bound name of the specified socket, store
37097 this address in the **sockaddr** structure pointed to by the `address` argument, and store the length of
37098 this address in the object pointed to by the `address_len` argument.

37099 The `address_len` argument points to a **socklen_t** object which on input specifies the length of the
37100 supplied **sockaddr** structure, and on output specifies the length of the stored address. If the
37101 actual length of the address is greater than the length of the supplied **sockaddr** structure, the
37102 stored address shall be truncated.

37103 If the socket has not been bound to a local name, the value stored in the object pointed to by
37104 `address` is unspecified.

37105 **RETURN VALUE**

37106 Upon successful completion, 0 shall be returned, the `address` argument shall point to the address
37107 of the socket, and the `address_len` argument shall point to the length of the address. Otherwise, -1
37108 shall be returned and `errno` set to indicate the error.

37109 **ERRORS**

37110 The `getsockname()` function shall fail if:

37111 [EBADF] The `socket` argument is not a valid file descriptor.

37112 [ENOTSOCK] The `socket` argument does not refer to a socket.

37113 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

37114 The `getsockname()` function may fail if:

37115 [EINVAL] The socket has been shut down.

37116 [ENOBUFS] Insufficient resources were available in the system to complete the function.

37117 **EXAMPLES**

37118 None.

37119 **APPLICATION USAGE**

37120 None.

37121 **RATIONALE**

37122 None.

37123 **FUTURE DIRECTIONS**

37124 None.

37125 **SEE ALSO**

37126 [*accept\(\)*](#), [*bind\(\)*](#), [*getpeername\(\)*](#), [*socket\(\)*](#)

37127 XBD [*<sys/socket.h>*](#)

37128 **CHANGE HISTORY**

37129 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37130 The **restrict** keyword is added to the *getsockname()* prototype for alignment with the
37131 ISO/IEC 9899:1999 standard.

37132 **Issue 7**

37133 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0259 [464] is applied.

37134 **NAME**

37135 getsockopt ¶get the socket options

37136 **SYNOPSIS**

```
37137 #include <sys/socket.h>
37138 int getsockopt(int socket, int level, int option_name,
37139 void *restrict option_value, socklen_t *restrict option_len);
```

37140 **DESCRIPTION**37141 The *getsockopt()* function manipulates options associated with a socket.

37142 The *getsockopt()* function shall retrieve the value for the option specified by the *option_name*
 37143 argument for the socket specified by the *socket* argument. If the size of the option value is greater
 37144 than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be
 37145 silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified
 37146 to indicate the actual length of the value.

37147 The *level* argument specifies the protocol level at which the option resides. To retrieve options at
 37148 the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels,
 37149 supply the appropriate level identifier for the protocol controlling the option. For example, to
 37150 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to
 37151 IPPROTO_TCP as defined in the **<netinet/in.h>** header.

37152 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*
 37153 function.

37154 The *option_name* argument specifies a single option to be retrieved. It can be one of the socket-
 37155 level options defined in **<sys/socket.h>** and described in [Section 2.10.16](#) (on page 528).

37156 **RETURN VALUE**

37157 Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*
 37158 set to indicate the error.

37159 **ERRORS**37160 The *getsockopt()* function shall fail if:

- 37161 [EBADF] The *socket* argument is not a valid file descriptor.
- 37162 [EINVAL] The specified option is invalid at the specified socket level.
- 37163 [ENOPROTOOPT]
- 37164 The option is not supported by the protocol.
- 37165 [ENOTSOCK] The *socket* argument does not refer to a socket.

37166 The *getsockopt()* function may fail if:

- 37167 [EACCES] The calling process does not have appropriate privileges.
- 37168 [EINVAL] The socket has been shut down.
- 37169 [ENOBUFS] Insufficient resources are available in the system to complete the function.

37170 **EXAMPLES**

37171 None.

37172 **APPLICATION USAGE**

37173 None.

37174 **RATIONALE**

37175 None.

37176 **FUTURE DIRECTIONS**

37177 None.

37178 **SEE ALSO**37179 [Section 2.10.16](#) (on page 528), [bind\(\)](#), [close\(\)](#), [endprotoent\(\)](#), [setsockopt\(\)](#), [socket\(\)](#)37180 XBD [<sys/socket.h>](#), [<netinet/in.h>](#)37181 **CHANGE HISTORY**

37182 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37183 The **restrict** keyword is added to the `getsockopt()` prototype for alignment with the
37184 ISO/IEC 9899:1999 standard.37185 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/47 is applied, updating the description of
37186 SO_LINGER in the DESCRIPTION so that it blocks the calling thread rather than the process.37187 **Issue 7**37188 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options
37189 that is now in [Section 2.10.16](#) (on page 528).

37190 **NAME**

37191 getsubopt — parse suboption arguments from a string

37192 **SYNOPSIS**

```
37193 CX #include <stdlib.h>
37194 int getsubopt(char **optionp, char * const *keylistp, char **valuep);
```

37195 **DESCRIPTION**

37196 The *getsubopt()* function shall parse suboption arguments in a flag argument. Such options often
 37197 result from the use of *getopt()*.

37198 The *getsubopt()* argument *optionp* is a pointer to a pointer to the option argument string. The
 37199 suboption arguments shall be separated by <comma> characters and each may consist of either
 37200 a single token, or a token-value pair separated by an <equals-sign>.

37201 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified
 37202 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in
 37203 **optionp*. Since <comma> characters delimit suboption arguments in *optionp*, they should not
 37204 appear in any of the strings pointed to by *keylistp*. Similarly, because an <equals-sign> separates
 37205 a token from its value, the application should not include an <equals-sign> in any of the strings
 37206 pointed to by *keylistp*. The *getsubopt()* function shall not modify the *keylistp* vector.

37207 The *valuep* argument is the address of a value string pointer.

37208 If a <comma> appears in *optionp*, it shall be interpreted as a suboption separator. After <comma>
 37209 characters have been processed, if there are one or more <equals-sign> characters in a suboption
 37210 string, the first <equals-sign> in any suboption string shall be interpreted as a separator between
 37211 a token and a value. Subsequent <equals-sign> characters in a suboption string shall be
 37212 interpreted as part of the value.

37213 If the string at **optionp* contains only one suboption argument (equivalently, no <comma>
 37214 characters), *getsubopt()* shall update **optionp* to point to the null character at the end of the
 37215 string. Otherwise, it shall isolate the suboption argument by replacing the <comma> separator
 37216 with a null character, and shall update **optionp* to point to the start of the next suboption
 37217 argument. If the suboption argument has an associated value (equivalently, contains an <equals-
 37218 sign>), *getsubopt()* shall update **valuep* to point to the value's first character. Otherwise, it shall
 37219 set **valuep* to a null pointer. The calling application may use this information to determine
 37220 whether the presence or absence of a value for the suboption is an error.

37221 Additionally, when *getsubopt()* fails to match the suboption argument with a token in the *keylistp*
 37222 array, the calling application should decide if this is an error, or if the unrecognized option
 37223 should be processed in another way.

37224 **RETURN VALUE**

37225 The *getsubopt()* function shall return the index of the matched token string, or -1 if no token
 37226 strings were matched.

37227 **ERRORS**

37228 No errors are defined.

37229 EXAMPLES

37230 Parsing Suboptions

37231 The following example uses the `getsubopt()` function to parse a *value* argument in the *optarg*
 37232 external variable returned by a call to `getopt()`.

```

37233 #include <stdio.h>
37234 #include <stdlib.h>
37235 #include <unistd.h>

37236 int do_all;
37237 const char *type;
37238 int read_size;
37239 int write_size;
37240 int read_only;

37241 enum
37242 {
37243     RO_OPTION = 0,
37244     RW_OPTION,
37245     READ_SIZE_OPTION,
37246     WRITE_SIZE_OPTION
37247 };

37248 const char *mount_opts[] =
37249 {
37250     [RO_OPTION] = "ro",
37251     [RW_OPTION] = "rw",
37252     [READ_SIZE_OPTION] = "rsize",
37253     [WRITE_SIZE_OPTION] = "wsize",
37254     NULL
37255 };

37256 int
37257 main(int argc, char *argv[])
37258 {
37259     char *subopts, *value;
37260     int opt;

37261     while ((opt = getopt(argc, argv, "at:o:")) != -1)
37262         switch(opt)
37263         {
37264             case 'a':
37265                 do_all = 1;
37266                 break;
37267             case 't':
37268                 type = optarg;
37269                 break;
37270             case 'o':
37271                 subopts = optarg;
37272                 while (*subopts != ' ')
37273                     {
37274                         char *saved = subopts;
37275                         switch(getsubopt(&subopts, (char **)mount_opts,
```

```

37276         &value))
37277     {
37278     case RO_OPTION:
37279         read_only = 1;
37280         break;
37281     case RW_OPTION:
37282         read_only = 0;
37283         break;
37284     case READ_SIZE_OPTION:
37285         if (value == NULL)
37286             abort();
37287         read_size = atoi(value);
37288         break;
37289     case WRITE_SIZE_OPTION:
37290         if (value == NULL)
37291             abort();
37292         write_size = atoi(value);
37293         break;
37294     default:
37295         /* Unknown suboption. */
37296         printf("Unknown suboption `%s'\n", saved);
37297         abort();
37298     }
37299     }
37300     break;
37301     default:
37302         abort();
37303     }
37304     /* Do the real work. */
37305     return 0;
37306 }

```

37307 If the above example is invoked with:

```
37308 program -o ro,rsize=512
```

37309 then after option parsing, the variable *do_all* will be 0, *type* will be a null pointer, *read_size* will be
37310 512, *write_size* will be 0, and *read_only* will be 1. If it is invoked with:

```
37311 program -o oops
```

37312 it will print:

```
37313 "Unknown suboption `oops'"
```

37314 before aborting.

37315 APPLICATION USAGE

37316 The value of **valuep* when *getsubopt()* returns *-1* is unspecified. Historical implementations
37317 provide various incompatible extensions to allow an application to access the suboption text that
37318 was not found in the *keylistp* array.

37319 **RATIONALE**

37320 The *keylistp* argument of *getsubopt()* is typed as **char * const *** to match historical practice.
37321 However, the standard is clear that implementations will not modify either the array or the
37322 strings contained in the array, as if the argument had been typed **const char * const ***.

37323 **FUTURE DIRECTIONS**

37324 None.

37325 **SEE ALSO**

37326 [getopt\(\)](#)

37327 XBD [<stdlib.h>](#)

37328 **CHANGE HISTORY**

37329 First released in Issue 4, Version 2.

37330 **Issue 5**

37331 Moved from X/OPEN UNIX extension to BASE.

37332 **Issue 6**

37333 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error
37334 in the SYNOPSIS.

37335 **Issue 7**

37336 The *getsubopt()* function is moved from the XSI option to the Base.

37337 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0260 [196], XSH/TC1-2008/0261 [196],
37338 XSH/TC1-2008/0262 [196], XSH/TC1-2008/0263 [196], XSH/TC1-2008/0264 [196],
37339 XSH/TC1-2008/0265 [196], XSH/TC1-2008/0266 [196], XSH/TC1-2008/0267 [196],
37340 XSH/TC1-2008/0268 [196], and XSH/TC1-2008/0269 [196] are applied.

37341 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0174 [791] is applied.

37342 **NAME**

37343 gettimeofday ‡'get the date and time

37344 **SYNOPSIS**

37345 OB XSI #include <sys/time.h>

37346 int gettimeofday(struct timeval *restrict tp, void *restrict tzp);

37347 **DESCRIPTION**37348 The *gettimeofday()* function shall obtain the current time, expressed as seconds and microseconds
37349 since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The resolution of the
37350 system clock is unspecified.37351 If *tzp* is not a null pointer, the behavior is unspecified.37352 **RETURN VALUE**37353 The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.37354 **ERRORS**

37355 No errors are defined.

37356 **EXAMPLES**

37357 None.

37358 **APPLICATION USAGE**37359 Applications should use the *clock_gettime()* function instead of the obsolescent *gettimeofday()*
37360 function.37361 **RATIONALE**

37362 None.

37363 **FUTURE DIRECTIONS**37364 The *gettimeofday()* function may be removed in a future version.37365 **SEE ALSO**37366 *clock_getres()*, *ctime()*

37367 XBD <sys/time.h>

37368 **CHANGE HISTORY**

37369 First released in Issue 4, Version 2.

37370 **Issue 5**

37371 Moved from X/OPEN UNIX extension to BASE.

37372 **Issue 6**37373 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
37374 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
37375 functions.37376 The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the
37377 ISO/IEC 9899:1999 standard.37378 **Issue 7**37379 The *gettimeofday()* function is marked obsolescent.

37380 **NAME**

37381 getuid — get a real user ID

37382 **SYNOPSIS**

37383 #include <unistd.h>

37384 uid_t getuid(void);

37385 **DESCRIPTION**37386 The *getuid()* function shall return the real user ID of the calling process. The *getuid()* function
37387 shall not modify *errno*.37388 **RETURN VALUE**37389 The *getuid()* function shall always be successful and no return value is reserved to indicate the
37390 error.37391 **ERRORS**

37392 No errors are defined.

37393 **EXAMPLES**37394 **Setting the Effective User ID to the Real User ID**

37395 The following example sets the effective user ID of the calling process to the real user ID.

37396 #include <unistd.h>

37397 ...

37398 seteuid(getuid());

37399 **APPLICATION USAGE**

37400 None.

37401 **RATIONALE**37402 In a conforming environment, *getuid()* will always succeed. It is possible for implementations to
37403 provide an extension where a process in a non-conforming environment will not be associated
37404 with a user or group ID. It is recommended that such implementations return **(uid_t)-1** and set
37405 *errno* to indicate such an environment; doing so does not violate this standard, since such an
37406 environment is already an extension.37407 **FUTURE DIRECTIONS**

37408 None.

37409 **SEE ALSO**37410 *getegid()*, *geteuid()*, *getgid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

37411 XBD <sys/types.h>, <unistd.h>

37412 **CHANGE HISTORY**

37413 First released in Issue 1. Derived from Issue 1 of the SVID.

37414 **Issue 6**

37415 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

37416 The following new requirements on POSIX implementations derive from alignment with the
37417 Single UNIX Specification:37418 The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
37419 required for conforming implementations of previous POSIX specifications, it was not
37420 required for UNIX applications.

37421 **Issue 7**

37422

37423

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0175 [511] and XSH/TC2-2008/0176 [897] are applied.

37424 **NAME**

37425 getutxent, getutxid, getutxline ‡get user accounting database entries

37426 **SYNOPSIS**

```
37427 XSI       #include <utmpx.h>
37428           struct utmpx *getutxent(void);
37429           struct utmpx *getutxid(const struct utmpx *id);
37430           struct utmpx *getutxline(const struct utmpx *line);
```

37431 **DESCRIPTION**

37432 Refer to *endutxent()*.

37433 **NAME**

37434 getwc — get a wide character from a stream

37435 **SYNOPSIS**

37436 #include <stdio.h>

37437 #include <wchar.h>

37438 wint_t getwc(FILE *stream);

37439 **DESCRIPTION**

37440 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37441 conflict between the requirements described here and the ISO C standard is unintentional. This
37442 volume of POSIX.1-2017 defers to the ISO C standard.

37443 The *getwc()* function shall be equivalent to *fgetwc()*, except that if it is implemented as a macro it
37444 may evaluate *stream* more than once, so the argument should never be an expression with side-
37445 effects.

37446 **RETURN VALUE**37447 Refer to *fgetwc()*.37448 **ERRORS**37449 Refer to *fgetwc()*.37450 **EXAMPLES**

37451 None.

37452 **APPLICATION USAGE**

37453 Since it may be implemented as a macro, *getwc()* may treat incorrectly a *stream* argument with
37454 side-effects. In particular, *getwc(*f++)* does not necessarily work as expected. Therefore, use of
37455 this function is not recommended; *fgetwc()* should be used instead.

37456 **RATIONALE**

37457 None.

37458 **FUTURE DIRECTIONS**

37459 None.

37460 **SEE ALSO**37461 [Section 2.5](#) (on page 495), *fgetwc()*37462 XBD [<stdio.h>](#), [<wchar.h>](#)37463 **CHANGE HISTORY**

37464 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
37465 draft.

37466 **Issue 5**37467 The Optional Header (OH) marking is removed from [<stdio.h>](#).37468 **Issue 7**

37469 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0270 [14] is applied.

37470 **NAME**37471 getwchar — get a wide character from a *stdin* stream37472 **SYNOPSIS**

37473 #include <wchar.h>

37474 wint_t getwchar(void);

37475 **DESCRIPTION**

37476 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37477 conflict between the requirements described here and the ISO C standard is unintentional. This
37478 volume of POSIX.1-2017 defers to the ISO C standard.

37479 The *getwchar()* function shall be equivalent to *getwc(stdin)*.37480 **RETURN VALUE**37481 Refer to *fgetwc()*.37482 **ERRORS**37483 Refer to *fgetwc()*.37484 **EXAMPLES**

37485 None.

37486 **APPLICATION USAGE**

37487 If the **wint_t** value returned by *getwchar()* is stored into a variable of type **wchar_t** and then
37488 compared against the **wint_t** macro WEOF, the result may be incorrect. Only the **wint_t** type is
37489 guaranteed to be able to represent any wide character and WEOF.

37490 **RATIONALE**

37491 None.

37492 **FUTURE DIRECTIONS**

37493 None.

37494 **SEE ALSO**37495 Section 2.5 (on page 495), *fgetwc()*, *getwc()*37496 XBD <**wchar.h**>37497 **CHANGE HISTORY**

37498 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
37499 draft.

37500 **Issue 7**

37501 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0271 [14] is applied.

37502 **NAME**

37503 glob, globfree — generate pathnames matching a pattern

37504 **SYNOPSIS**

```
37505 #include <glob.h>
37506 int glob(const char *restrict pattern, int flags,
37507         int(*errfunc)(const char *epath, int eerrno),
37508         glob_t *restrict pglob);
37509 void globfree(glob_t *pglob);
```

37510 **DESCRIPTION**

37511 The *glob()* function is a pathname generator that shall implement the rules defined in XCU
 37512 [Section 2.13](#) (on page 2382), with optional support for rule 3 in XCU [Section 2.13.3](#) (on page
 37513 2383).

37514 The structure type **glob_t** is defined in **<glob.h>** and includes at least the following members:

Member Type	Member Name	Description
size_t	<i>gl_pathc</i>	Count of paths matched by <i>pattern</i> .
char **	<i>gl_pathv</i>	Pointer to a list of matched pathnames.
size_t	<i>gl_offs</i>	Slots to reserve at the beginning of <i>gl_pathv</i> .

37520 The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function
 37521 shall match all accessible pathnames against this pattern and develop a list of all pathnames that
 37522 match. In order to have access to a pathname, *glob()* requires search permission on every
 37523 component of a path except the last, and read permission on each directory of any filename
 37524 component of *pattern* that contains any of the following special characters: '*', '?', and '['.

37525 The *glob()* function shall store the number of matched pathnames into *pglob* *gl_pathc* and a
 37526 pointer to a list of pointers to pathnames into *pglob* *gl_pathv*. The pathnames shall be in sort
 37527 order as defined by the current setting of the *LC_COLLATE* category; see XBD [Section 7.3.2](#) (on
 37528 page 147). The first pointer after the last pathname shall be a null pointer. If the pattern does not
 37529 match any pathnames, the returned number of matched paths is set to 0, and the contents of
 37530 *pglob* *gl_pathv* are implementation-defined.

37531 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function
 37532 shall allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree()*
 37533 function shall free any space associated with *pglob* from a previous call to *glob()*.

37534 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-
 37535 inclusive OR of zero or more of the following constants, which are defined in **<glob.h>**:

37536	GLOB_APPEND	Append pathnames generated to the ones from a previous call to <i>glob()</i> .
37537	GLOB_DOOFFS	Make use of <i>pglob</i> <i>gl_offs</i> . If this flag is set, <i>pglob</i> <i>gl_offs</i> is used to 37538 specify how many null pointers to add to the beginning of 37539 <i>pglob</i> <i>gl_pathv</i> . In other words, <i>pglob</i> <i>gl_pathv</i> shall point to 37540 <i>pglob</i> <i>gl_offs</i> null pointers, followed by <i>pglob</i> <i>gl_pathc</i> pathname 37541 pointers, followed by a null pointer.
37542	GLOB_ERR	Cause <i>glob()</i> to return when it encounters a directory that it cannot open 37543 or read. Ordinarily, <i>glob()</i> continues to find matches.
37544	GLOB_MARK	Each pathname that is a directory that matches <i>pattern</i> shall have a 37545 <slash> appended.

- 37546 GLOB_NOCHECK Supports rule 3 in XCU [Section 2.13.3](#) (on page 2383). If *pattern* does not
 37547 match any pathname, then *glob()* shall return a list consisting of only
 37548 *pattern*, and the number of matched pathnames is 1.
- 37549 GLOB_NOESCAPE Disable backslash escaping.
- 37550 GLOB_NOSORT Ordinarily, *glob()* sorts the matching pathnames according to the current
 37551 setting of the *LC_COLLATE* category; see XBD [Section 7.3.2](#) (on page 147).
 37552 When this flag is used, the order of pathnames returned is unspecified.

37553 The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a
 37554 previous call to *glob()*. The following rules apply to applications when two or more calls to
 37555 *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- 37556 1. The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
- 37557 2. All the calls shall set GLOB_DOOFFS, or all shall not set it.
- 37558 3. After the second call, *pglob* *gl_pathv* points to a list containing the following:
 - 37559 a. Zero or more null pointers, as specified by GLOB_DOOFFS and *pglob* *gl_offs*.
 - 37560 b. Pointers to the pathnames that were in the *pglob* *gl_pathv* list before the call, in
 37561 the same order as before.
 - 37562 c. Pointers to the new pathnames generated by the second call, in the specified order.
- 37563 4. The count returned in *pglob* *gl_pathc* shall be the total number of pathnames from the
 37564 two calls.
- 37565 5. The application can change any of the fields after a call to *glob()*. If it does, the
 37566 application shall reset them to the original value before a subsequent call, using the same
 37567 *pglob* value, to *globfree()* or *glob()* with the GLOB_APPEND flag.

37568 If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not
 37569 a null pointer, *glob()* calls (**errfunc()*) with two arguments:

- 37570 1. The *epath* argument is a pointer to the path that failed.
- 37571 2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or
 37572 *stat()*. (Other values may be used to report other errors not explicitly documented for
 37573 those functions.)

37574 If (**errfunc()*) is called and returns non-zero, or if the GLOB_ERR flag is set in *flags*, *glob()* shall
 37575 stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect
 37576 the paths already scanned. If GLOB_ERR is not set and either *errfunc* is a null pointer or
 37577 (**errfunc()*) returns 0, the error shall be ignored.

37578 The *glob()* function shall not fail because of large files.

37579 RETURN VALUE

37580 Upon successful completion, *glob()* shall return 0. The argument *pglob* *gl_pathc* shall return the
 37581 number of matched pathnames and the argument *pglob* *gl_pathv* shall contain a pointer to a
 37582 null-terminated list of matched and sorted pathnames. However, if *pglob* *gl_pathc* is 0, the
 37583 content of *pglob* *gl_pathv* is undefined.

37584 The *globfree()* function shall not return a value.

37585 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in
 37586 **<glob.h>**. The arguments *pglob* *gl_pathc* and *pglob* *gl_pathv* are still set as defined above.

37587 **ERRORS**

37588 The *glob()* function shall fail and return the corresponding value if:

37589 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (**errfunc()*)
37590 returned non-zero.

37591 GLOB_NOMATCH The pattern does not match any existing pathname, and
37592 GLOB_NOCHECK was not set in flags.

37593 GLOB_NOSPACE An attempt to allocate memory failed.

37594 **EXAMPLES**

37595 One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with
37596 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the
37597 equivalent of:

```
37598 ls -l *.c
```

37599 but for some reason:

```
37600 system("ls -l *.c")
```

37601 is not acceptable. The application could obtain approximately the same result using the
37602 sequence:

```
37603 globbuf.gl_offs = 2;
37604 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
37605 globbuf.gl_pathv[0] = "ls";
37606 globbuf.gl_pathv[1] = "-l";
37607 execvp("ls", &globbuf.gl_pathv[0]);
```

37608 Using the same example:

```
37609 ls -l *.c *.h
```

37610 could be approximately simulated using GLOB_APPEND as follows:

```
37611 globbuf.gl_offs = 2;
37612 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
37613 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
37614 ...
```

37615 **APPLICATION USAGE**

37616 This function is not provided for the purpose of enabling utilities to perform pathname
37617 expansion on their arguments, as this operation is performed by the shell, and utilities are
37618 explicitly not expected to redo this. Instead, it is provided for applications that need to do
37619 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or
37620 read from a file.

37621 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

37622 Note that *gl_pathc* and *gl_pathv* have meaning even if *glob()* fails. This allows *glob()* to report
37623 partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if
37624 *glob()* did not return an error.

37625 The GLOB_NOCHECK option could be used when an application wants to expand a pathname
37626 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility
37627 might use this for option-arguments, for example.

37628 The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted
37629 together with the previous pathnames. This mirrors the way that the shell handles pathname

37630 expansion when multiple expansions are done on a command line.

37631 Applications that need tilde and parameter expansion should use *wordexp()*.

37632 RATIONALE

37633 It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
37634 new = (char **)malloc((n + pglob->gl_pathc + 1)
37635     * sizeof(char *));
37636 (void) memcpy(new+n, pglob->gl_pathv,
37637     pglob->gl_pathc * sizeof(char *));
37638 (void) memset(new, 0, n * sizeof(char *));
37639 free(pglob->gl_pathv);
37640 pglob->gl_pathv = new;
```

37641 However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately
 37642 created using *malloc()*. This is not necessarily the case. An application should make no
 37643 assumptions about how the memory referenced by fields in *pglob* was allocated. It might have
 37644 been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have
 37645 been created using a different memory allocator. It is not the intent of the standard developers to
 37646 specify or imply how the memory used by *glob()* is managed.

37647 The GLOB_APPEND flag would be used when an application wants to expand several different
 37648 patterns into a single list.

37649 FUTURE DIRECTIONS

37650 None.

37651 SEE ALSO

37652 [exec](#), [fdopendir\(\)](#), [fnmatch\(\)](#), [fstatat\(\)](#), [readdir\(\)](#), [Section 2.6](#)

37653 XBD [Section 7.3.2](#) (on page 147), [<glob.h>](#)

37654 CHANGE HISTORY

37655 First released in Issue 4. Derived from the ISO POSIX-2 standard.

37656 Issue 5

37657 Moved from POSIX2 C-language Binding to BASE.

37658 Issue 6

37659 The normative text is updated to avoid use of the term “must” for application requirements.

37660 The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999
 37661 standard.

37662 **NAME**

37663 gmtime, gmtime_r — convert a time value to a broken-down UTC time

37664 **SYNOPSIS**

```
37665 #include <time.h>
37666 struct tm *gmtime(const time_t *timer);
37667 CX struct tm *gmtime_r(const time_t *restrict timer,
37668 struct tm *restrict result);
```

37669 **DESCRIPTION**

37670 CX For `gmtime()`: The functionality described on this reference page is aligned with the ISO C
 37671 standard. Any conflict between the requirements described here and the ISO C standard is
 37672 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

37673 The `gmtime()` function shall convert the time in seconds since the Epoch pointed to by `timer` into
 37674 a broken-down time, expressed as Coordinated Universal Time (UTC).

37675 CX The relationship between a time in seconds since the Epoch used as an argument to `gmtime()`
 37676 and the `tm` structure (defined in the `<time.h>` header) is that the result shall be as specified in
 37677 the expression given in the definition of seconds since the Epoch (see XBD Section 4.16, on page
 37678 113), where the names in the structure and in the expression correspond.

37679 The same relationship shall apply for `gmtime_r()`.

37680 The `gmtime()` function need not be thread-safe.

37681 The `asctime()`, `ctime()`, `gmtime()`, and `localtime()` functions shall return values in one of two static
 37682 objects: a broken-down time structure and an array of type `char`. Execution of any of the
 37683 functions may overwrite the information returned in either of these objects by any of the other
 37684 functions.

37685 The `gmtime_r()` function shall convert the time in seconds since the Epoch pointed to by `timer`
 37686 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down
 37687 time is stored in the structure referred to by `result`. The `gmtime_r()` function shall also return the
 37688 address of the same structure.

37689 **RETURN VALUE**

37690 Upon successful completion, the `gmtime()` function shall return a pointer to a `struct tm`. If an
 37691 CX error is detected, `gmtime()` shall return a null pointer and set `errno` to indicate the error.

37692 Upon successful completion, `gmtime_r()` shall return the address of the structure pointed to by
 37693 the argument `result`. If an error is detected, `gmtime_r()` shall return a null pointer and set `errno` to
 37694 indicate the error.

37695 **ERRORS**

37696 CX The `gmtime()` and `gmtime_r()` functions shall fail if:

37697 CX [EOVERFLOW] The result cannot be represented.

37698 **EXAMPLES**

37699 None.

37700 **APPLICATION USAGE**

37701 The *gmtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of
37702 possibly using a static data area that may be overwritten by each call.

37703 **RATIONALE**

37704 None.

37705 **FUTURE DIRECTIONS**

37706 None.

37707 **SEE ALSO**37708 *asctime()*, *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*, *utime()*37709 XBD Section 4.16 (on page 113), <[time.h](#)>37710 **CHANGE HISTORY**

37711 First released in Issue 1. Derived from Issue 1 of the SVID.

37712 **Issue 5**

37713 A note indicating that the *gmtime()* function need not be reentrant is added to the
37714 DESCRIPTION.

37715 The *gmtime_r()* function is included for alignment with the POSIX Threads Extension.37716 **Issue 6**37717 The *gmtime_r()* function is marked as part of the Thread-Safe Functions option.

37718 Extensions beyond the ISO C standard are marked.

37719 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
37720 its avoidance of possibly using a static data area.

37721 The **restrict** keyword is added to the *gmtime_r()* prototype for alignment with the
37722 ISO/IEC 9899:1999 standard.

37723 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [EOVERFLOW]
37724 error.

37725 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/48 is applied, updating the error handling
37726 for *gmtime_r()*.

37727 **Issue 7**

37728 Austin Group Interpretation 1003.1-2001 #156 is applied.

37729 The *gmtime_r()* function is moved from the Thread-Safe Functions option to the Base.

37730 **NAME**

37731 grantpt ‡grant access to the slave pseudo-terminal device

37732 **SYNOPSIS**

```
37733 XSI #include <stdlib.h>
37734 int grantpt(int fildev);
```

37735 **DESCRIPTION**

37736 The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal
 37737 device associated with its master pseudo-terminal counterpart. The *fildev* argument is a file
 37738 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to
 37739 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The
 37740 permission mode of the slave pseudo-terminal shall be set to readable and writable by the
 37741 owner, and writable by the group.

37742 The behavior of the *grantpt()* function is unspecified if the application has installed a signal
 37743 handler to catch SIGCHLD signals.

37744 **RETURN VALUE**

37745 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to
 37746 indicate the error.

37747 **ERRORS**

37748 The *grantpt()* function may fail if:

37749 [EACCES] The corresponding slave pseudo-terminal device could not be accessed.

37750 [EBADF] The *fildev* argument is not a valid open file descriptor.

37751 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.

37752 **EXAMPLES**

37753 None.

37754 **APPLICATION USAGE**

37755 None.

37756 **RATIONALE**

37757 See the RATIONALE section for *posix_openpt()*.

37758 **FUTURE DIRECTIONS**

37759 None.

37760 **SEE ALSO**

37761 *open()*, *posix_openpt()*, *ptsname()*, *unlockpt()*

37762 XBD <stdlib.h>

37763 **CHANGE HISTORY**

37764 First released in Issue 4, Version 2.

37765 **Issue 5**

37766 Moved from X/OPEN UNIX extension to BASE.

37767 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

37768 **Issue 7**

37769 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0272 [96] is applied.

37770 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0177 [506] is applied.

37771 **NAME**

37772 hcreate, hdestroy, hsearch — manage hash search table

37773 **SYNOPSIS**

```
37774 XSI #include <search.h>
37775
37775 int hcreate(size_t nel);
37776 void hdestroy(void);
37777 ENTRY *hsearch(ENTRY item, ACTION action);
```

37778 **DESCRIPTION**37779 The *hcreate()*, *hdestroy()*, and *hsearch()* functions shall manage hash search tables.

37780 The *hcreate()* function shall allocate sufficient space for the table, and the application shall ensure it is called before *hsearch()* is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

37784 The *hdestroy()* function shall dispose of the search table, and may be followed by another call to *hcreate()*. After the call to *hdestroy()*, the data can no longer be considered accessible.

37786 The *hsearch()* function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the **<search.h>** header) containing two pointers: *item.key* points to the comparison key (a **char ***), and *item.data* (a **void ***) points to any other data to be associated with that key. The comparison function used by *hsearch()* is *strcmp()*. The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

37795 These functions need not be thread-safe.

37796 **RETURN VALUE**37797 The *hcreate()* function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.37799 The *hdestroy()* function shall not return a value.37800 The *hsearch()* function shall return a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.37802 **ERRORS**37803 The *hcreate()* and *hsearch()* functions may fail if:

37804 [ENOMEM] Insufficient storage space is available.

37805 **EXAMPLES**

37806 The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
37809 #include <stdio.h>
37810 #include <search.h>
37811 #include <string.h>
37812
37812 struct info {          /* This is the info stored in the table */
37813     int age, room;     /* other than the key. */
37814 };
```

```

37815     #define NUM_EMPL      5000    /* # of elements in search table. */
37816     int main(void)
37817     {
37818         char string_space[NUM_EMPL*20]; /* Space to store strings. */
37819         struct info info_space[NUM_EMPL]; /* Space to store employee info. */
37820         char *str_ptr = string_space; /* Next space in string_space. */
37821         struct info *info_ptr = info_space;
37822                                     /* Next space in info_space. */
37823         ENTRY item;
37824         ENTRY *found_item; /* Name to look for in table. */
37825         char name_to_find[30];
37826
37827         int i = 0;
37828
37829         /* Create table; no error checking is performed. */
37830         (void) hcreate(NUM_EMPL);
37831         while (scanf("%s%d%d", str_ptr, &info_ptr->age,
37832                     &info_ptr->room) != EOF && i++ < NUM_EMPL) {
37833
37834             /* Put information in structure, and structure in item. */
37835             item.key = str_ptr;
37836             item.data = info_ptr;
37837             str_ptr += strlen(str_ptr) + 1;
37838             info_ptr++;
37839
37840             /* Put item into table. */
37841             (void) hsearch(item, ENTER);
37842         }
37843
37844         /* Access table. */
37845         item.key = name_to_find;
37846         while (scanf("%s", item.key) != EOF) {
37847             if ((found_item = hsearch(item, FIND)) != NULL) {
37848                 /* If item is in the table. */
37849                 (void)printf("found %s, age = %d, room = %d\n",
37850                             found_item->key,
37851                             ((struct info *)found_item->data)->age,
37852                             ((struct info *)found_item->data)->room);
37853             } else
37854                 (void)printf("no such employee %s\n", name_to_find);
37855         }
37856         return 0;
37857     }

```

APPLICATION USAGE

The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

RATIONALE

None.

FUTURE DIRECTIONS

None.

37859 **SEE ALSO**37860 *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tdelete()*37861 XBD <[search.h](#)>37862 **CHANGE HISTORY**

37863 First released in Issue 1. Derived from Issue 1 of the SVID.

37864 **Issue 6**

37865 The normative text is updated to avoid use of the term “must” for application requirements.

37866 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

37867 **Issue 7**

37868 Austin Group Interpretation 1003.1-2001 #156 is applied.

37869 **NAME**

37870 htonl, htons, ntohl, ntohs — convert values between host and network byte order

37871 **SYNOPSIS**

```
37872     #include <arpa/inet.h>
37873     uint32_t htonl(uint32_t hostlong);
37874     uint16_t htons(uint16_t hostshort);
37875     uint32_t ntohl(uint32_t netlong);
37876     uint16_t ntohs(uint16_t netshort);
```

37877 **DESCRIPTION**37878 These functions shall convert 16-bit and 32-bit quantities between network byte order and host
37879 byte order.

37880 On some implementations, these functions are defined as macros.

37881 The `uint32_t` and `uint16_t` types are defined in `<inttypes.h>`.37882 **RETURN VALUE**37883 The `htonl()` and `htons()` functions shall return the argument value converted from host to
37884 network byte order.37885 The `ntohl()` and `ntohs()` functions shall return the argument value converted from network to
37886 host byte order.37887 **ERRORS**

37888 No errors are defined.

37889 **EXAMPLES**

37890 None.

37891 **APPLICATION USAGE**37892 These functions are most often used in conjunction with IPv4 addresses and ports as returned by
37893 `gethostent()` and `getservent()`.37894 **RATIONALE**

37895 None.

37896 **FUTURE DIRECTIONS**

37897 None.

37898 **SEE ALSO**37899 [*endhostent\(\)*](#), [*endservent\(\)*](#)37900 XBD [`<arpa/inet.h>`](#), [`<inttypes.h>`](#)37901 **CHANGE HISTORY**

37902 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

37903 **NAME**

37904 hypot, hypotf, hypotl ‡Euclidean distance function

37905 **SYNOPSIS**

37906 #include <math.h>

37907 double hypot(double x, double y);

37908 float hypotf(float x, float y);

37909 long double hypotl(long double x, long double y);

37910 **DESCRIPTION**

37911 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37912 conflict between the requirements described here and the ISO C standard is unintentional. This
 37913 volume of POSIX.1-2017 defers to the ISO C standard.

37914 These functions shall compute the value of the square root of x^2+y^2 without undue overflow or
 37915 underflow.

37916 An application wishing to check for error situations should set *errno* to zero and call
 37917 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 37918 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 37919 zero, an error has occurred.

37920 **RETURN VALUE**

37921 Upon successful completion, these functions shall return the length of the hypotenuse of a right-
 37922 angled triangle with sides of length *x* and *y*.

37923 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and
 37924 *hypotl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 37925 respectively.

37926 MX If *x* or *y* is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned (even if one of *x* or *y* is NaN).

37927 If *x* or *y* is NaN, and the other is not $\pm\text{Inf}$, a NaN shall be returned.

37928 MXX If both arguments are subnormal and the correct result is subnormal, a range error may occur
 37929 and the correct result shall be returned.

37930 **ERRORS**

37931 These functions shall fail if:

37932 Range Error The result overflows.

37933 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 37934 then *errno* shall be set to [ERANGE]. If the integer expression
 37935 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 37936 floating-point exception shall be raised.

37937 These functions may fail if:

37938 MX Range Error The result underflows.

37939 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 37940 then *errno* shall be set to [ERANGE]. If the integer expression
 37941 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 37942 floating-point exception shall be raised.

37943 **EXAMPLES**37944 See the EXAMPLES section in *atan2()*.37945 **APPLICATION USAGE**37946 *hypot(x,y)*, *hypot(y,x)*, and *hypot(x,-y)* are equivalent.37947 *hypot(x, ±0)* is equivalent to *fabs(x)*.37948 Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also
37949 subnormal.

37950 These functions take precautions against overflow during intermediate steps of the computation.

37951 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
37952 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.37953 **RATIONALE**

37954 None.

37955 **FUTURE DIRECTIONS**

37956 None.

37957 **SEE ALSO**37958 *atan2()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*

37959 XBD Section 4.20 (on page 117), <math.h>

37960 **CHANGE HISTORY**

37961 First released in Issue 1. Derived from Issue 1 of the SVID.

37962 **Issue 5**37963 The DESCRIPTION is updated to indicate how an application should check for an error. This
37964 text was previously published in the APPLICATION USAGE section.37965 **Issue 6**37966 The *hypot()* function is no longer marked as an extension.37967 The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999
37968 standard.37969 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
37970 revised to align with the ISO/IEC 9899:1999 standard.37971 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
37972 marked.37973 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/49 is applied, updating the EXAMPLES
37974 section.37975 **Issue 7**

37976 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0273 [68] is applied.

37977 **NAME**

37978 iconv ‡codeset conversion function

37979 **SYNOPSIS**

```
37980 #include <iconv.h>
37981 size_t iconv(iconv_t cd, char **restrict inbuf,
37982             size_t *restrict inbytesleft, char **restrict outbuf,
37983             size_t *restrict outbytesleft);
```

37984 **DESCRIPTION**

37985 The *iconv()* function shall convert the sequence of characters from one codeset, in the array
 37986 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array
 37987 specified by *outbuf*. The codesets are those specified in the *iconv_open()* call that returned the
 37988 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first
 37989 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer
 37990 to be converted. The *outbuf* argument points to a variable that points to the first available byte in
 37991 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the
 37992 buffer.

37993 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by
 37994 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()* is
 37995 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*
 37996 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change
 37997 the output buffer to its initial shift state. If the output buffer is not large enough to hold the
 37998 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as
 37999 other than a null pointer or a pointer to a null pointer cause the conversion to take place from
 38000 the current state of the conversion descriptor.

38001 If a sequence of input bytes does not form a valid character in the specified codeset, conversion
 38002 shall stop after the previous successfully converted character. If the input buffer ends with an
 38003 incomplete character or shift sequence, conversion shall stop after the previous successfully
 38004 converted bytes. If the output buffer is not large enough to hold the entire converted input,
 38005 conversion shall stop just prior to the input bytes that would cause the output buffer to
 38006 overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last
 38007 byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be
 38008 decremented to reflect the number of bytes still not converted in the input buffer. The variable
 38009 pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted
 38010 output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of
 38011 bytes still available in the output buffer. For state-dependent encodings, the conversion
 38012 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully
 38013 converted byte sequence.

38014 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical
 38015 character does not exist in the target codeset, *iconv()* shall perform an implementation-defined
 38016 conversion on this character.

38017 **RETURN VALUE**

38018 The *iconv()* function shall update the variables pointed to by the arguments to reflect the extent
 38019 of the conversion and return the number of non-identical conversions performed. If the entire
 38020 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input
 38021 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*
 38022 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs, *iconv()* shall
 38023 return (**size_t**)-1 and set *errno* to indicate the error.

38024 **ERRORS**38025 The *iconv()* function shall fail if:38026 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the
38027 input codeset.

38028 [E2BIG] Input conversion stopped due to lack of space in the output buffer.

38029 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at
38030 the end of the input buffer.38031 The *iconv()* function may fail if:38032 [EBADF] The *cd* argument is not a valid open conversion descriptor.38033 **EXAMPLES**

38034 None.

38035 **APPLICATION USAGE**38036 The *inbuf* argument indirectly points to the memory area which contains the conversion input
38037 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of
38038 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to
38039 containing data that is directly representable in the ISO C standard language **char** data type. The
38040 type of *inbuf* and *outbuf*, **char ****, does not imply that the objects pointed to are interpreted as
38041 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that
38042 represents a character in a given character set encoding scheme is done internally within the
38043 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can
38044 contain all zero octets that are not interpreted as string terminators but as coded character data
38045 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and
38046 so on) read or stored in the objects is not specified, but may be inferred for both the input and
38047 output data by the converters determined by the *fromcode* and *toencode* arguments of *iconv_open()*.38048 Regardless of the data type inferred by the converter, the size of the remaining space in both
38049 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.38050 For implementations that support the conversion of state-dependent encodings, the conversion
38051 descriptor must be able to accurately reflect the shift-state in effect at the end of the last
38052 successful conversion. It is not required that the conversion descriptor itself be updated, which
38053 would require it to be a pointer type. Thus, implementations are free to implement the
38054 descriptor as a handle (other than a pointer type) by which the conversion information can be
38055 accessed and updated.38056 **RATIONALE**

38057 None.

38058 **FUTURE DIRECTIONS**

38059 None.

38060 **SEE ALSO**38061 [iconv_open\(\)](#), [iconv_close\(\)](#), [mbsrtowcs\(\)](#)38062 XBD [<iconv.h>](#)38063 **CHANGE HISTORY**

38064 First released in Issue 4. Derived from the HP-UX Manual.

38065 **Issue 6**

38066 The SYNOPSIS has been corrected to align with the `<iconv.h>` reference page.

38067 The **restrict** keyword is added to the `iconv()` prototype for alignment with the
38068 ISO/IEC 9899:1999 standard.

38069 **Issue 7**

38070 The `iconv()` function is moved from the XSI option to the Base.

38071 **NAME**

38072 iconv_close ‡codeset conversion deallocation function

38073 **SYNOPSIS**

38074 #include <iconv.h>

38075 int iconv_close(iconv_t cd);

38076 **DESCRIPTION**38077 The *iconv_close()* function shall deallocate the conversion descriptor *cd* and all other associated
38078 resources allocated by *iconv_open()*.38079 If a file descriptor is used to implement the type **iconv_t**, that file descriptor shall be closed.38080 **RETURN VALUE**38081 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
38082 indicate the error.38083 **ERRORS**38084 The *iconv_close()* function may fail if:

38085 [EBADF] The conversion descriptor is invalid.

38086 **EXAMPLES**

38087 None.

38088 **APPLICATION USAGE**

38089 None.

38090 **RATIONALE**

38091 None.

38092 **FUTURE DIRECTIONS**

38093 None.

38094 **SEE ALSO**38095 *iconv()*, *iconv_open()*38096 XBD <[iconv.h](#)>38097 **CHANGE HISTORY**

38098 First released in Issue 4. Derived from the HP-UX Manual.

38099 **Issue 7**38100 The *iconv_close()* function is moved from the XSI option to the Base.

38101 **NAME**

38102 iconv_open †'codeset conversion allocation function

38103 **SYNOPSIS**

38104 #include <iconv.h>

38105 iconv_t iconv_open(const char **tocode*, const char **fromcode*);38106 **DESCRIPTION**

38107 The *iconv_open()* function shall return a conversion descriptor that describes a conversion from
38108 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified
38109 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion
38110 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with
38111 *iconv()*.

38112 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.38113 A conversion descriptor shall remain valid until it is closed by *iconv_close()* or an implicit close.

38114 If a file descriptor is used to implement conversion descriptors, the FD_CLOEXEC flag shall be
38115 set; see <fcntl.h>.

38116 **RETURN VALUE**

38117 Upon successful completion, *iconv_open()* shall return a conversion descriptor for use on
38118 subsequent calls to *iconv()*. Otherwise, *iconv_open()* shall return (**iconv_t**)-1 and set *errno* to
38119 indicate the error.

38120 **ERRORS**38121 The *iconv_open()* function may fail if:

38122 [EMFILE] All file descriptors available to the process are currently open.

38123 [ENFILE] Too many files are currently open in the system.

38124 [ENOMEM] Insufficient storage space is available.

38125 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the
38126 implementation.

38127 **EXAMPLES**

38128 None.

38129 **APPLICATION USAGE**

38130 Some implementations of *iconv_open()* use *malloc()* to allocate space for internal buffer areas.
38131 The *iconv_open()* function may fail if there is insufficient storage space to accommodate these
38132 buffers.

38133 Conforming applications must assume that conversion descriptors are not valid after a call to
38134 one of the *exec* functions.

38135 Application developers should consult the system documentation to determine the supported
38136 codesets and their naming schemes.

38137 **RATIONALE**

38138 None.

38139 **FUTURE DIRECTIONS**

38140 None.

38141 **SEE ALSO**

38142 [iconv\(\)](#), [iconv_close\(\)](#)

38143 XBD [<fcntl.h>](#), [<iconv.h>](#)

38144 **CHANGE HISTORY**

38145 First released in Issue 4. Derived from the HP-UX Manual.

38146 **Issue 7**

38147 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

38148 The *iconv_open()* function is moved from the XSI option to the Base.

38149 NAME

38150 if_freenameindex — free memory allocated by if_nameindex

38151 SYNOPSIS

38152 #include <net/if.h>

38153 void if_freenameindex(struct if_nameindex *ptr);

38154 DESCRIPTION

38155 The *if_freenameindex()* function shall free the memory allocated by *if_nameindex()*. The *ptr*
38156 argument shall be a pointer that was returned by *if_nameindex()*. After *if_freenameindex()* has
38157 been called, the application shall not use the array of which *ptr* is the address.

38158 RETURN VALUE

38159 None.

38160 ERRORS

38161 No errors are defined.

38162 EXAMPLES

38163 None.

38164 APPLICATION USAGE

38165 None.

38166 RATIONALE

38167 None.

38168 FUTURE DIRECTIONS

38169 None.

38170 SEE ALSO

38171 *getsockopt()*, *if_indextoname()*, *if_nameindex()*, *if_nametoindex()*, *setsockopt()*

38172 XBD <net/if.h>

38173 CHANGE HISTORY

38174 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

38175 NAME

38176 `if_indextoname` — map a network interface index to its corresponding name

38177 SYNOPSIS

38178 `#include <net/if.h>`

38179 `char *if_indextoname(unsigned ifindex, char *ifname);`

38180 DESCRIPTION

38181 The `if_indextoname()` function shall map an interface index to its corresponding name.

38182 When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The
38183 function shall place in this buffer the name of the interface with index *ifindex*.

38184 RETURN VALUE

38185 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which
38186 points to a buffer now containing the interface name. Otherwise, the function shall return a null
38187 pointer and set *errno* to indicate the error.

38188 ERRORS

38189 The `if_indextoname()` function shall fail if:

38190 [ENXIO] The interface does not exist.

38191 EXAMPLES

38192 None.

38193 APPLICATION USAGE

38194 None.

38195 RATIONALE

38196 None.

38197 FUTURE DIRECTIONS

38198 None.

38199 SEE ALSO

38200 [`getsockopt\(\)`](#), [`if_freenameindex\(\)`](#), [`if_nameindex\(\)`](#), [`if_nametoindex\(\)`](#), [`setsockopt\(\)`](#)

38201 XBD [`<net/if.h>`](#)

38202 CHANGE HISTORY

38203 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

38204 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to
38205 {IF_NAMESIZ} in the DESCRIPTION.

38206 NAME

38207 if_nameindex — return all network interface names and indexes

38208 SYNOPSIS

```
38209 #include <net/if.h>
38210 struct if_nameindex *if_nameindex(void);
```

38211 DESCRIPTION

38212 The *if_nameindex()* function shall return an array of *if_nameindex* structures, one structure per
38213 interface. The end of the array is indicated by a structure with an *if_index* field of zero and an
38214 *if_name* field of NULL.

38215 Applications should call *if_freenameindex()* to release the memory that may be dynamically
38216 allocated by this function, after they have finished using it.

38217 RETURN VALUE

38218 An array of structures identifying local interfaces. A null pointer is returned upon an error, with
38219 *errno* set to indicate the error.

38220 ERRORS

38221 The *if_nameindex()* function may fail if:

38222 [ENOBUFS] Insufficient resources are available to complete the function.

38223 EXAMPLES

38224 None.

38225 APPLICATION USAGE

38226 None.

38227 RATIONALE

38228 None.

38229 FUTURE DIRECTIONS

38230 None.

38231 SEE ALSO

38232 [*getsockopt\(\)*](#), [*if_freenameindex\(\)*](#), [*if_indextoname\(\)*](#), [*if_nametoindex\(\)*](#), [*setsockopt\(\)*](#)

38233 XBD [**<net/if.h>**](#)

38234 CHANGE HISTORY

38235 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

38236 **NAME**

38237 if_nametoindex — map a network interface name to its corresponding index

38238 **SYNOPSIS**

38239 #include <net/if.h>

38240 unsigned if_nametoindex(const char *ifname);

38241 **DESCRIPTION**

38242 The *if_nametoindex()* function shall return the interface index corresponding to name *ifname*.

38243 **RETURN VALUE**

38244 The corresponding index if *ifname* is the name of an interface; otherwise, zero.

38245 **ERRORS**

38246 No errors are defined.

38247 **EXAMPLES**

38248 None.

38249 **APPLICATION USAGE**

38250 None.

38251 **RATIONALE**

38252 None.

38253 **FUTURE DIRECTIONS**

38254 None.

38255 **SEE ALSO**

38256 *getsockopt()*, *if_freenameindex()*, *if_indextoname()*, *if_nameindex()*, *setsockopt()*

38257 XBD <net/if.h>

38258 **CHANGE HISTORY**

38259 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

38260 **NAME**38261 `ilogb`, `ilogbf`, `ilogbl` — return an unbiased exponent38262 **SYNOPSIS**

```
38263     #include <math.h>
38264     int ilogb(double x);
38265     int ilogbf(float x);
38266     int ilogbl(long double x);
```

38267 **DESCRIPTION**

38268 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 38269 conflict between the requirements described here and the ISO C standard is unintentional. This
 38270 volume of POSIX.1-2017 defers to the ISO C standard.

38271 These functions shall return the exponent part of their argument x . Formally, the return value is
 38272 the integral part of $\log_r |x|$ as a signed integral value, for non-zero x , where r is the radix of the
 38273 machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

38274 An application wishing to check for error situations should set `errno` to zero and call
 38275 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 38276 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 38277 zero, an error has occurred.

38278 **RETURN VALUE**

38279 Upon successful completion, these functions shall return the exponent part of x as a signed
 38280 integer value. They are equivalent to calling the corresponding `logb()` function and casting the
 38281 returned value to type `int`.

38282 XSI If x is 0, the value `FP_ILOGB0` shall be returned. On XSI-conformant systems, a domain error
 38283 shall occur;

38284 CX otherwise, a domain error may occur.

38285 XSI If x is $\pm\text{Inf}$, the value `{INT_MAX}` shall be returned. On XSI-conformant systems, a domain
 38286 error shall occur;

38287 CX otherwise, a domain error may occur.

38288 XSI If x is a NaN, the value `FP_ILOGBNAN` shall be returned. On XSI-conformant systems, a
 38289 domain error shall occur;

38290 CX otherwise, a domain error may occur.

38291 MX If the correct value is greater than `{INT_MAX}`, a domain error shall occur and an unspecified
 38292 XSI value shall be returned. On XSI-conformant systems, a domain error shall occur and
 38293 `{INT_MAX}` shall be returned.

38294 MX If the correct value is less than `{INT_MIN}`, a domain error shall occur and an unspecified value
 38295 XSI shall be returned. On XSI-conformant systems, a domain error shall occur and `{INT_MIN}` shall
 38296 be returned.

38297 **ERRORS**

38298 These functions shall fail if:

38299 XSI|MX **Domain Error** The correct value is not representable as an integer.

38300 XSI The x argument is zero, NaN, or $\pm\text{Inf}$.

38301 XSI|MX If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,
 38302 then `errno` shall be set to `[EDOM]`. If the integer expression (`math_errhandling`
 38303 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception
 38304 shall be raised.

38305 These functions may fail if:

38306 Domain Error The x argument is zero, NaN, or $\pm\text{Inf}$.

38307 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 38308 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 38309 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 38310 shall be raised.

38311 EXAMPLES

38312 None.

38313 APPLICATION USAGE

38314 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 38315 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

38316 RATIONALE

38317 The errors come from taking the expected floating-point value and converting it to **int**, which is
 38318 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not
 38319 representable in a type **int**), so should be a domain error.

38320 There are no known implementations that overflow. For overflow to happen, {INT_MAX} must
 38321 be less than $\text{LDBL_MAX_EXP} \cdot \log_2(\text{FLT_RADIX})$ or {INT_MIN} must be greater than
 38322 $\text{LDBL_MIN_EXP} \cdot \log_2(\text{FLT_RADIX})$ if subnormals are not supported, or {INT_MIN} must be
 38323 greater than $(\text{LDBL_MIN_EXP} - \text{LDBL_MANT_DIG}) \cdot \log_2(\text{FLT_RADIX})$ if subnormals are
 38324 supported.

38325 FUTURE DIRECTIONS

38326 None.

38327 SEE ALSO

38328 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [logb\(\)](#), [scalbln\(\)](#)

38329 XBD Section 4.20 (on page 117), [<float.h>](#), [<math.h>](#)

38330 CHANGE HISTORY

38331 First released in Issue 4, Version 2.

38332 Issue 5

38333 Moved from X/OPEN UNIX extension to BASE.

38334 Issue 6

38335 The *ilogb()* function is no longer marked as an extension.

38336 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999
 38337 standard.

38338 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

38339 Functionality relating to the XSI option is marked.

38340 Issue 7

38341 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79
 38342 (SD5-XSH-ERN-72) are applied.

38343 **NAME**

38344 imaxabs — return absolute value

38345 **SYNOPSIS**

38346 #include <inttypes.h>

38347 intmax_t imaxabs(intmax_t j);

38348 **DESCRIPTION**

38349 CX The functionality described on this reference page is aligned with the ISO C standard. Any
38350 conflict between the requirements described here and the ISO C standard is unintentional. This
38351 volume of POSIX.1-2017 defers to the ISO C standard.

38352 The *imaxabs()* function shall compute the absolute value of an integer *j*. If the result cannot be
38353 represented, the behavior is undefined.

38354 **RETURN VALUE**38355 The *imaxabs()* function shall return the absolute value.38356 **ERRORS**

38357 No errors are defined.

38358 **EXAMPLES**

38359 None.

38360 **APPLICATION USAGE**

38361 The absolute value of the most negative number cannot be represented in two's complement.

38362 **RATIONALE**

38363 None.

38364 **FUTURE DIRECTIONS**

38365 None.

38366 **SEE ALSO**38367 [imaxdiv\(\)](#)38368 XBD [<inttypes.h>](#)38369 **CHANGE HISTORY**

38370 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38371 **NAME**

38372 imaxdiv — return quotient and remainder

38373 **SYNOPSIS**

38374 #include <inttypes.h>

38375 imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);

38376 **DESCRIPTION**

38377 CX The functionality described on this reference page is aligned with the ISO C standard. Any
38378 conflict between the requirements described here and the ISO C standard is unintentional. This
38379 volume of POSIX.1-2017 defers to the ISO C standard.

38380 The *imaxdiv()* function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.38381 **RETURN VALUE**

38382 The *imaxdiv()* function shall return a structure of type **imaxdiv_t**, comprising both the quotient
38383 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)
38384 and *rem* (the remainder), each of which has type **intmax_t**.

38385 If either part of the result cannot be represented, the behavior is undefined.

38386 **ERRORS**

38387 No errors are defined.

38388 **EXAMPLES**

38389 None.

38390 **APPLICATION USAGE**

38391 None.

38392 **RATIONALE**

38393 None.

38394 **FUTURE DIRECTIONS**

38395 None.

38396 **SEE ALSO**38397 [imaxabs\(\)](#)38398 XBD [<inttypes.h>](#)38399 **CHANGE HISTORY**

38400 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

38401 **NAME**38402 `inet_addr, inet_ntoa` — IPv4 address manipulation38403 **SYNOPSIS**

```
38404 #include <arpa/inet.h>
38405 in_addr_t inet_addr(const char *cp);
38406 char *inet_ntoa(struct in_addr in);
```

38407 **DESCRIPTION**

38408 The `inet_addr()` function shall convert the string pointed to by `cp`, in the standard IPv4 dotted
38409 decimal notation, to an integer value suitable for use as an Internet address.

38410 The `inet_ntoa()` function shall convert the Internet host address specified by `in` to a string in the
38411 Internet standard dot notation.

38412 The `inet_ntoa()` function need not be thread-safe.

38413 All Internet addresses shall be returned in network order (bytes ordered from left to right).

38414 Values specified using IPv4 dotted decimal notation take one of the following forms:

38415 `a.b.c.d` When four parts are specified, each shall be interpreted as a byte of data and
38416 assigned, from left to right, to the four bytes of an Internet address.

38417 `a.b.c` When a three-part address is specified, the last part shall be interpreted as a 16-bit
38418 quantity and placed in the rightmost two bytes of the network address. This makes
38419 the three-part address format convenient for specifying Class B network addresses
38420 as "`128.net.host`".

38421 `a.b` When a two-part address is supplied, the last part shall be interpreted as a 24-bit
38422 quantity and placed in the rightmost three bytes of the network address. This
38423 makes the two-part address format convenient for specifying Class A network
38424 addresses as "`net.host`".

38425 `a` When only one part is given, the value shall be stored directly in the network
38426 address without any byte rearrangement.

38427 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or
38428 hexadecimal, as specified in the ISO C standard (that is, a leading `0x` or `0X` implies hexadecimal;
38429 otherwise, a leading `'0'` implies octal; otherwise, the number is interpreted as decimal).

38430 **RETURN VALUE**

38431 Upon successful completion, `inet_addr()` shall return the Internet address. Otherwise, it shall
38432 return (`in_addr_t`)(-1).

38433 The `inet_ntoa()` function shall return a pointer to the network address in Internet standard dot
38434 notation.

38435 **ERRORS**

38436 No errors are defined.

38437 EXAMPLES

38438 None.

38439 APPLICATION USAGE

38440 The return value of *inet_ntoa()* may point to static data that may be overwritten by subsequent
38441 calls to *inet_ntoa()*.

38442 RATIONALE

38443 None.

38444 FUTURE DIRECTIONS

38445 None.

38446 SEE ALSO

38447 *endhostent()*, *endnetent()*

38448 XBD <[arpa/inet.h](#)>

38449 CHANGE HISTORY

38450 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

38451 Issue 7

38452 Austin Group Interpretation 1003.1-2001 #156 is applied.

38453 **NAME**38454 `inet_ntop, inet_pton` — convert IPv4 and IPv6 addresses between binary and text form38455 **SYNOPSIS**

```
38456 #include <arpa/inet.h>
38457
38457 const char *inet_ntop(int af, const void *restrict src,
38458 char *restrict dst, socklen_t size);
38459 int inet_pton(int af, const char *restrict src, void *restrict dst);
```

38460 **DESCRIPTION**

38461 The `inet_ntop()` function shall convert a numeric address into a text string suitable for
 38462 IP6 presentation. The `af` argument shall specify the family of the address. This can be `AF_INET` or
 38463 `AF_INET6`. The `src` argument points to a buffer holding an IPv4 address if the `af` argument is
 38464 IP6 `AF_INET`, or an IPv6 address if the `af` argument is `AF_INET6`; the address must be in network
 38465 byte order. The `dst` argument points to a buffer where the function stores the resulting text string;
 38466 it shall not be NULL. The `size` argument specifies the size of this buffer, which shall be large
 38467 IP6 enough to hold the text string (`INET_ADDRSTRLEN` characters for IPv4,
 38468 `INET6_ADDRSTRLEN` characters for IPv6).

38469 The `inet_pton()` function shall convert an address in its standard text presentation form into its
 38470 IP6 numeric binary form. The `af` argument shall specify the family of the address. The `AF_INET` and
 38471 `AF_INET6` address families shall be supported. The `src` argument points to the string being
 38472 passed in. The `dst` argument points to a buffer into which the function stores the numeric
 38473 IP6 address; this shall be large enough to hold the numeric address (32 bits for `AF_INET`, 128 bits
 38474 for `AF_INET6`).

38475 If the `af` argument of `inet_pton()` is `AF_INET`, the `src` string shall be in the standard IPv4 dotted-
 38476 decimal form:

```
38477 ddd.ddd.ddd.ddd
```

38478 where "ddd" is a one to three digit decimal number between 0 and 255 (see `inet_addr()`). The
 38479 `inet_pton()` function does not accept other formats (such as the octal numbers, hexadecimal
 38480 numbers, and fewer than four numbers that `inet_addr()` accepts).

38481 IP6 If the `af` argument of `inet_pton()` is `AF_INET6`, the `src` string shall be in one of the following
 38482 standard IPv6 text forms:

- 38483 1. The preferred form is "`x:x:x:x:x:x:x:x`", where the 'x's are the hexadecimal values
 38484 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be
 38485 omitted, but there shall be one to four hexadecimal digits in every field.
- 38486 2. A string of contiguous zero fields in the preferred form can be shown as "`::`". The "`::`"
 38487 can only appear once in an address. Unspecified addresses ("`0:0:0:0:0:0:0:0`") may
 38488 be represented simply as "`::`".
- 38489 3. A third form that is sometimes more convenient when dealing with a mixed environment
 38490 of IPv4 and IPv6 nodes is "`x:x:x:x:x:x.d.d.d.d`", where the 'x's are the
 38491 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the
 38492 decimal values of the four low-order 8-bit pieces of the address (standard IPv4
 38493 representation).

38494 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in
 38495 RFC 2373.

38496 **RETURN VALUE**

38497 The *inet_ntop()* function shall return a pointer to the buffer containing the text string if the
38498 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

38499 The *inet_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by
38500 IP6 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string
38501 or a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is
38502 unknown.

38503 **ERRORS**

38504 The *inet_ntop()* and *inet_pton()* functions shall fail if:

38505 [EAFNOSUPPORT]

38506 The *af* argument is invalid.

38507 [ENOSPC] The size of the *inet_ntop()* result buffer is inadequate.

38508 **EXAMPLES**

38509 None.

38510 **APPLICATION USAGE**

38511 None.

38512 **RATIONALE**

38513 None.

38514 **FUTURE DIRECTIONS**

38515 None.

38516 **SEE ALSO**

38517 XBD <[arpa/inet.h](#)>

38518 **CHANGE HISTORY**

38519 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

38520 IPv6 extensions are marked.

38521 The **restrict** keyword is added to the *inet_ntop()* and *inet_pton()* prototypes for alignment with
38522 the ISO/IEC 9899:1999 standard.

38523 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding “the address must
38524 be in network byte order” to the end of the fourth sentence of the first paragraph in the
38525 DESCRIPTION.

38526 **Issue 7**

38527 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0178 [777] is applied.

38528 **NAME**

38529 initstate, random, setstate, srandom ‡pseudo-random number functions

38530 **SYNOPSIS**

```
38531 XSI       #include <stdlib.h>
38532       char *initstate(unsigned seed, char *state, size_t size);
38533       long random(void);
38534       char *setstate(char *state);
38535       void srandom(unsigned seed);
```

38536 **DESCRIPTION**

38537 The *random()* function shall use a non-linear additive feedback random-number generator
 38538 employing a default state array size of 31 **long** integers to return successive pseudo-random
 38539 numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is
 38540 approximately $16 \times (2^{31}-1)$. The size of the state array determines the period of the random-
 38541 number generator. Increasing the state array size shall increase the period.

38542 With 256 bytes of state information, the period of the random-number generator shall be greater
 38543 than 2^{69} .

38544 Like *rand()*, *random()* shall produce by default a sequence of numbers that can be duplicated by
 38545 calling *srandom()* with 1 as the seed.

38546 The *srandom()* function shall initialize the current state array using the value of *seed*.

38547 The *initstate()* and *setstate()* functions handle restarting and changing random-number
 38548 generators. The *initstate()* function allows a state array, pointed to by the *state* argument, to be
 38549 initialized for future use. The *size* argument, which specifies the size in bytes of the state array,
 38550 shall be used by *initstate()* to decide what type of random-number generator to use; the larger
 38551 the state array, the more random the numbers. Values for the amount of state information are 8,
 38552 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one
 38553 of these values. If *initstate()* is called with $8 \leq \text{size} < 32$, then *random()* shall use a simple linear
 38554 congruential random number generator. The *seed* argument specifies a starting point for the
 38555 random-number sequence and provides for restarting at the same point. The *initstate()* function
 38556 shall return a pointer to the previous state information array.

38557 If *initstate()* has not been called, then *random()* shall behave as though *initstate()* had been called
 38558 with *seed*=1 and *size*=128.

38559 Once a state has been initialized, *setstate()* allows switching between state arrays. The array
 38560 defined by the *state* argument shall be used for further random-number generation until
 38561 *initstate()* is called or *setstate()* is called again. The *setstate()* function shall return a pointer to the
 38562 previous state array.

38563 **RETURN VALUE**

38564 If *initstate()* is called with *size* less than 8, it shall return NULL.

38565 The *random()* function shall return the generated pseudo-random number.

38566 The *srandom()* function shall not return a value.

38567 Upon successful completion, *initstate()* and *setstate()* shall return a pointer to the previous state
 38568 array; otherwise, a null pointer shall be returned.

38569 **ERRORS**

38570 No errors are defined.

38571 **EXAMPLES**

38572 None.

38573 **APPLICATION USAGE**

38574 After initialization, a state array can be restarted at a different point in one of two ways:

- 38575 1. The *initstate()* function can be used, with the desired seed, state array, and size of the
38576 array.
- 38577 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with
38578 the desired seed. The advantage of using both of these functions is that the size of the
38579 state array does not have to be saved once it is initialized.

38580 Although some implementations of *random()* have written messages to standard error, such
38581 implementations do not conform to POSIX.1-2017.

38582 Issue 5 restored the historical behavior of this function.

38583 Threaded applications should use *erand48()*, *nrand48()*, or *jrand48()* instead of *random()* when
38584 an independent random number sequence in multiple threads is required.

38585 These functions should be avoided whenever non-trivial requirements (including safety) have to
38586 be fulfilled.

38587 **RATIONALE**

38588 None.

38589 **FUTURE DIRECTIONS**

38590 None.

38591 **SEE ALSO**38592 [*drand48\(\)*, *rand\(\)*](#)38593 XBD [`<stdlib.h>`](#)38594 **CHANGE HISTORY**

38595 First released in Issue 4, Version 2.

38596 **Issue 5**

38597 Moved from X/OPEN UNIX extension to BASE.

38598 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than
38599 or equal to 8, or less than 32”, “size<8” is replaced with “8≤size <32”, and a new first paragraph
38600 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE
38601 indicating that these changes restore the historical behavior of the function.

38602 **Issue 6**

38603 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.

38604 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand_r()* from the
38605 list of suggested functions in the APPLICATION USAGE section.

38606 **Issue 7**38607 The type of the first argument to *setstate()* is changed from **const char *** to **char ***.

38608 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0179 [743] is applied.

38609 **NAME**

38610 insque, remque — insert or remove an element in a queue

38611 **SYNOPSIS**

```
38612 XSI #include <search.h>
38613 void insque(void *element, void *pred);
38614 void remque(void *element);
```

38615 **DESCRIPTION**

38616 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The
 38617 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it
 38618 defines a structure in which the first two members of the structure are pointers to the same type
 38619 of structure, and any further members are application-specific. The first member of the structure
 38620 is a forward pointer to the next entry in the queue. The second member is a backward pointer to
 38621 the previous entry in the queue. If the queue is linear, the queue is terminated with null
 38622 pointers. The names of the structure and of the pointer members are not subject to any special
 38623 restriction.

38624 The *insque()* function shall insert the element pointed to by *element* into a queue immediately
 38625 after the element pointed to by *pred*.

38626 The *remque()* function shall remove the element pointed to by *element* from a queue.

38627 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the
 38628 initial element of the queue, shall initialize the forward and backward pointers of *element* to null
 38629 pointers.

38630 If the queue is to be used as a circular list, the application shall ensure it initializes the forward
 38631 pointer and the backward pointer of the initial element of the queue to the element's own
 38632 address.

38633 **RETURN VALUE**38634 The *insque()* and *remque()* functions do not return a value.38635 **ERRORS**

38636 No errors are defined.

38637 **EXAMPLES**38638 **Creating a Linear Linked List**

38639 The following example creates a linear linked list.

```
38640 #include <search.h>
38641 ...
38642 struct myque element1;
38643 struct myque element2;
38644
38645 char *data1 = "DATA1";
38646 char *data2 = "DATA2";
38647 ...
38648 element1.data = data1;
38649 element2.data = data2;
38650
38651 insque (&element1, NULL);
38652 insque (&element2, &element1);
```

Creating a Circular Linked List

The following example creates a circular linked list.

```

38651 #include <search.h>
38652 ...
38653 struct myque element1;
38654 struct myque element2;
38655
38656 char *data1 = "DATA1";
38657 char *data2 = "DATA2";
38658 ...
38659 element1.data = data1;
38660 element2.data = data2;
38661
38662 element1.fwd = &element1;
38663 element1.bck = &element1;
38664 insque (&element2, &element1);

```

Removing an Element

The following example removes the element pointed to by *element1*.

```

38667 #include <search.h>
38668 ...
38669 struct myque element1;
38670 ...
38671 remque (&element1);

```

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type **struct qelem *** rather than as being of type **void *** as defined here. In those implementations, **struct qelem** was commonly defined in **<search.h>** as:

```

38676 struct qelem {
38677     struct qelem *q_forw;
38678     struct qelem *q_back;
38679 };

```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object's data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same name, if they literally used **struct qelem** as specified.

As described here, the implementations were actually expecting a structure type where the first two members were forward and backward pointers to structures. With C compilers that didn't provide function prototypes, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.

If this method had been carried forward with an ISO C standard compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By specifying **void *** as the argument type, applications do not need to change (unless they specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

38695 **RATIONALE**

38696 None.

38697 **FUTURE DIRECTIONS**

38698 None.

38699 **SEE ALSO**

38700 XBD [<search.h>](#)

38701 **CHANGE HISTORY**

38702 First released in Issue 4, Version 2.

38703 **Issue 5**

38704 Moved from X/OPEN UNIX extension to BASE.

38705 **Issue 6**

38706 The normative text is updated to avoid use of the term “must” for application requirements.

38707 **NAME**38708 ioctl — control a STREAMS device (**STREAMS**)38709 **SYNOPSIS**

```
38710 OB XSR #include <stropts.h>
38711 int ioctl(int fildev, int request, ... /* arg */);
```

38712 **DESCRIPTION**

38713 The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non-
 38714 STREAMS devices, the functions performed by this call are unspecified. The *request* argument
 38715 and an optional third argument (with varying type) shall be passed to and interpreted by the
 38716 appropriate part of the STREAM associated with *fildev*.

38717 The *fildev* argument is an open file descriptor that refers to a device.

38718 The *request* argument selects the control function to be performed and shall depend on the
 38719 STREAMS device being addressed.

38720 The *arg* argument represents additional information that is needed by this specific STREAMS
 38721 device to perform the requested function. The type of *arg* depends upon the particular control
 38722 request, but it shall be either an integer or a pointer to a device-specific data structure.

38723 The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply
 38724 to each individual command are described below.

38725 The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS
 38726 files:

38727 **I_PUSH** Pushes the module whose name is pointed to by *arg* onto the top of the current
 38728 STREAM, just below the STREAM head. It then calls the *open()* function of the
 38729 newly-pushed module.

38730 The *ioctl()* function with the I_PUSH command shall fail if:

38731 [EINVAL] Invalid module name.
 38732 [ENXIO] Open function of new module failed.
 38733 [ENXIO] Hangup received on *fildev*.

38734 **I_POP** Removes the module just below the STREAM head of the STREAM pointed to
 38735 by *fildev*. The *arg* argument should be 0 in an I_POP request.

38736 The *ioctl()* function with the I_POP command shall fail if:

38737 [EINVAL] No module present in the STREAM.
 38738 [ENXIO] Hangup received on *fildev*.

38739 **I_LOOK** Retrieves the name of the module just below the STREAM head of the
 38740 STREAM pointed to by *fildev*, and places it in a character string pointed to by
 38741 *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,
 38742 where FMNAMESZ is defined in **<stropts.h>**.

38743 The *ioctl()* function with the I_LOOK command shall fail if:

38744 [EINVAL] No module present in the STREAM.

38745 **I_FLUSH** Flushes read and/or write queues, depending on the value of *arg*. Valid *arg*
 38746 values are:

38747	FLUSHR	Flush all read queues.
38748	FLUSHW	Flush all write queues.
38749	FLUSHRW	Flush all read and all write queues.
38750	The <i>ioctl()</i> function with the I_FLUSH command shall fail if:	
38751	[EINVAL]	Invalid <i>arg</i> value.
38752	[EAGAIN] or [ENOSR]	Unable to allocate buffers for flush message.
38753		
38754	[ENXIO]	Hangup received on <i>fildev</i> .
38755	I_FLUSHBAND	Flushes a particular band of messages. The <i>arg</i> argument points to a bandinfo structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The <i>bi_pri</i> member determines the priority band to be flushed.
38756		
38757		
38758		
38759	I_SETSIG	Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants:
38760		
38761		
38762		
38763		
38764		
38765	S_RDNORM	
38766		
38767		
38768	S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
38769		
38770		
38771	S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
38772		
38773		
38774	S_HIPRI	A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.
38775		
38776		
38777	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
38778		
38779		
38780		
38781	S_WRNORM	Equivalent to S_OUTPUT.
38782	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
38783		
38784		
38785		
38786	S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
38787		
38788		

38789		S_ERROR	Notification of an error condition has reached the STREAM head.
38790			
38791		S_HANGUP	Notification of a hangup has reached the STREAM head.
38792		S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
38793			
38794			
38795			If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fildev</i> .
38796			
38797			Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.
38798			
38799			
38800			
38801			The <i>ioctl()</i> function with the I_SETSIG command shall fail if:
38802		[EINVAL]	The value of <i>arg</i> is invalid.
38803		[EINVAL]	The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.
38804			
38805		[EAGAIN]	There were insufficient resources to store the signal request.
38806	I_GETSIG		Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an int pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.
38807			
38808			
38809			
38810			The <i>ioctl()</i> function with the I_GETSIG command shall fail if:
38811		[EINVAL]	Process is not registered to receive the SIGPOLL signal.
38812	I_FIND		Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.
38813			
38814			
38815			The <i>ioctl()</i> function with the I_FIND command shall fail if:
38816		[EINVAL]	<i>arg</i> does not contain a valid module name.
38817	I_PEEK		Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a strpeek structure.
38818			
38819			
38820			
38821			The application shall ensure that the <i>maxlen</i> member in the ctlbuf and databuf structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.
38822			
38823			
38824			
38825			
38826			
38827			I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, ctlbuf specifies information in the control buffer, databuf specifies information in the data
38828			
38829			
38830			
38831			

38832		buffer, and <i>flags</i> contains the value RS_HIPRI or 0.
38833	I_SRDOPT	Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read()</i> . Valid <i>arg</i> flags are:
38834		
38835		RNORM Byte-stream mode, the default.
38836		RMSGD Message-discard mode.
38837		RMSGN Message-nondiscard mode.
38838		The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in the other flag overriding RNORM which is the default.
38839		
38840		
38841		In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in <i>arg</i> :
38842		
38843	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
38844		
38845	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
38846		
38847	RPROTDIS	Discard the control part of a message, delivering any data portion, when a process issues a <i>read()</i> .
38848		
38849		The <i>ioctl()</i> function with the I_SRDOPT command shall fail if:
38850		[EINVAL] The <i>arg</i> argument is not valid.
38851	I_GRDOPT	Returns the current read mode setting, as described above, in an int pointed to by the argument <i>arg</i> . Read modes are described in <i>read()</i> .
38852		
38853	I_NREAD	Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the int pointed to by <i>arg</i> . The return value for the command shall be the number of messages on the STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl()</i> return value is greater than 0, this indicates that a zero-length message is next on the queue.
38854		
38855		
38856		
38857		
38858		
38859	I_FDINSERT	Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The <i>arg</i> argument points to a strfdinsert structure.
38860		
38861		
38862		
38863		
38864		The application shall ensure that the <i>len</i> member in the ctlbuf strbuf structure is set to the size of a t_uscalar_t plus the number of bytes of control information to be sent with the message. The <i>fildev</i> member specifies the file descriptor of the other STREAM, and the <i>offset</i> member, which must be suitably aligned for use as a t_uscalar_t , specifies the offset from the start of the control buffer where I_FDINSERT shall store a t_uscalar_t whose interpretation is specific to the STREAM end. The application shall ensure that the <i>len</i> member in the databuf strbuf structure is set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.
38865		
38866		
38867		
38868		
38869		
38870		
38871		
38872		
38873		
38874		The <i>flags</i> member specifies the type of message to be created. A normal message is created if <i>flags</i> is set to 0, and a high-priority message is created if
38875		

38876 *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if
38877 the STREAM write queue is full due to internal flow control conditions. For
38878 priority messages, I_FDINSERT does not block on this condition. For non-
38879 priority messages, I_FDINSERT does not block when the write queue is full
38880 and O_NONBLOCK is set. Instead, it fails and sets *errno* to [EAGAIN].

38881 I_FDINSERT also blocks, unless prevented by lack of internal resources,
38882 waiting for the availability of message blocks in the STREAM, regardless of
38883 priority or whether O_NONBLOCK has been specified. No partial message is
38884 sent.

38885 The *ioctl()* function with the I_FDINSERT command shall fail if:

38886 [EAGAIN] A non-priority message is specified, the O_NONBLOCK
38887 flag is set, and the STREAM write queue is full due to
38888 internal flow control conditions.

38889 [EAGAIN] or [ENOSR]
38890 Buffers cannot be allocated for the message that is to be
38891 created.

38892 [EINVAL] One of the following:

38893 ‡ *filedes* member of the **strfdinsert** structure is not a
38894 valid, open STREAM file descriptor.

38895 ‡ *size* of a **t_uscalar_t** plus *offset* is greater than the
38896 *len* member for the buffer specified through **ctlbuf**.

38897 ‡ *offset* member does not specify a properly-aligned
38898 location in the data buffer.

38899 ‡ *undefined* value is stored in *flags*.

38900 [ENXIO] Hangup received on the STREAM identified by either the
38901 *filedes* argument or the *filedes* member of the **strfdinsert**
38902 structure.

38903 [ERANGE] The *len* member for the buffer specified through **databuf**
38904 does not fall within the range specified by the maximum
38905 and minimum packet sizes of the topmost STREAM
38906 module; or the *len* member for the buffer specified through
38907 **databuf** is larger than the maximum configured size of the
38908 data part of a message; or the *len* member for the buffer
38909 specified through **ctlbuf** is larger than the maximum
38910 configured size of the control part of a message.

38911 I_STR Constructs an internal STREAMS *ioctl()* message from the data pointed to by
38912 *arg*, and sends that message downstream.

38913 This mechanism is provided to send *ioctl()* requests to downstream modules
38914 and drivers. It allows information to be sent with *ioctl()*, and returns to the
38915 process any information sent upstream by the downstream recipient. I_STR
38916 shall block until the system responds with either a positive or negative
38917 acknowledgement message, or until the request times out after some period of
38918 time. If the request times out, it shall fail with *errno* set to [ETIME].

38919 At most, one I_STR can be active on a STREAM. Further I_STR calls shall
38920 block until the active I_STR completes at the STREAM head. The default

38921 timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has
 38922 no effect on this call.

38923 To send requests downstream, the application shall ensure that *arg* points to a
 38924 **striocctl** structure.

38925 The *ic_cmd* member is the internal *ioctl()* command intended for a
 38926 downstream module or driver and *ic_timeout* is the number of seconds
 38927 (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified)
 38928 an I_STR request shall wait for acknowledgement before timing out. *ic_len* is
 38929 the number of bytes in the data argument, and *ic_dp* is a pointer to the data
 38930 argument. The *ic_len* member has two uses: on input, it contains the length of
 38931 the data argument passed in, and on return from the command, it contains the
 38932 number of bytes being returned to the process (the buffer pointed to by *ic_dp*
 38933 should be large enough to contain the maximum amount of data that any
 38934 module or the driver in the STREAM can return).

38935 The STREAM head shall convert the information pointed to by the **striocctl**
 38936 structure to an internal *ioctl()* command message and send it downstream.

38937 The *ioctl()* function with the I_STR command shall fail if:

38938 [EAGAIN] or [ENOSR]
 38939 Unable to allocate buffers for the *ioctl()* message.

38940 [EINVAL] The *ic_len* member is less than 0 or larger than the
 38941 maximum configured size of the data part of a message, or
 38942 *ic_timeout* is less than −1.

38943 [ENXIO] Hangup received on *fildev*.

38944 [ETIME] A downstream *ioctl()* timed out before acknowledgement
 38945 was received.

38946 An I_STR can also fail while waiting for an acknowledgement if a message
 38947 indicating an error or a hangup is received at the STREAM head. In addition,
 38948 an error code can be returned in the positive or negative acknowledgement
 38949 message, in the event the *ioctl()* command sent downstream fails. For these
 38950 cases, I_STR shall fail with *errno* set to the value in the message.

38951 **I_SWROPT** Sets the write mode using the value of the argument *arg*. Valid bit settings for
 38952 *arg* are:

38953 SNDZERO Send a zero-length message downstream when a *write()* of 0
 38954 bytes occurs. To not send a zero-length message when a
 38955 *write()* of 0 bytes occurs, the application shall ensure that
 38956 this bit is not set in *arg* (for example, *arg* would be set to 0).

38957 The *ioctl()* function with the I_SWROPT command shall fail if:

38958 [EINVAL] *arg* is not the above value.

38959 **I_GWROPT** Returns the current write mode setting, as described above, in the **int** that is
 38960 pointed to by the argument *arg*.

38961 **I_SENDFD** Creates a new reference to the open file description associated with the file
 38962 descriptor *arg*, and writes a message on the STREAMS-based pipe *fildev*
 38963 containing this reference, together with the user ID and group ID of the calling
 38964 process.

38965 The *ioctl()* function with the `I_SENDFD` command shall fail if:

38966 [EAGAIN] The sending STREAM is unable to allocate a message block
38967 to contain the file pointer; or the read queue of the receiving
38968 STREAM head is full and cannot accept the message sent by
38969 `I_SENDFD`.

38970 [EBADF] The *arg* argument is not a valid, open file descriptor.

38971 [EINVAL] The *fildev* argument is not connected to a STREAM pipe.

38972 [ENXIO] Hangup received on *fildev*.

38973 The *ioctl()* function with the `I_SENDFD` command may fail if:

38974 [EINVAL] The *arg* argument is equal to the *fildev* argument.

38975 `I_RECVFD` Retrieves the reference to an open file description from a message written to a
38976 STREAMS-based pipe using the `I_SENDFD` command, and allocates a new file
38977 descriptor in the calling process that refers to this open file description. The
38978 *arg* argument is a pointer to a **strrecvfd** data structure as defined in
38979 **<stropts.h>**.

38980 The *fd* member is a file descriptor. The *uid* and *gid* members are the effective
38981 user ID and effective group ID, respectively, of the sending process.

38982 If `O_NONBLOCK` is not set, `I_RECVFD` shall block until a message is present
38983 at the STREAM head. If `O_NONBLOCK` is set, `I_RECVFD` shall fail with *errno*
38984 set to [EAGAIN] if no message is present at the STREAM head.

38985 If the message at the STREAM head is a message sent by an `I_SENDFD`, a new
38986 file descriptor shall be allocated for the open file descriptor referenced in the
38987 message. The new file descriptor is placed in the *fd* member of the **strrecvfd**
38988 structure pointed to by *arg*.

38989 The *ioctl()* function with the `I_RECVFD` command shall fail if:

38990 [EAGAIN] A message is not present at the STREAM head read queue
38991 and the `O_NONBLOCK` flag is set.

38992 [EBADMSG] The message at the STREAM head read queue is not a
38993 message containing a passed file descriptor.

38994 [EMFILE] All file descriptors available to the process are currently
38995 open.

38996 [ENXIO] Hangup received on *fildev*.

38997 `I_LIST` Allows the process to list all the module names on the STREAM, up to and
38998 including the topmost driver name. If *arg* is a null pointer, the return value
38999 shall be the number of modules, including the driver, that are on the STREAM
39000 pointed to by *fildev*. This lets the process allocate enough space for the module
39001 names. Otherwise, it should point to a **str_list** structure.

39002 The *sl_nmods* member indicates the number of entries the process has
39003 allocated in the array. Upon return, the *sl_modlist* member of the **str_list**
39004 structure shall contain the list of module names, and the number of entries
39005 that have been filled into the *sl_modlist* array is found in the *sl_nmods* member
39006 (the number includes the number of modules including the driver). The return
39007 value from *ioctl()* shall be 0. The entries are filled in starting at the top of the

39008		STREAM and continuing downstream until either the end of the STREAM is
39009		reached, or the number of requested modules (<i>sl_nmods</i>) is satisfied.
39010		The <i>ioctl()</i> function with the <code>I_LIST</code> command shall fail if:
39011		[EINVAL] The <i>sl_nmods</i> member is less than 1.
39012		[EAGAIN] or [ENOSR]
39013		Unable to allocate buffers.
39014	<code>I_ATMARK</code>	Allows the process to see if the message at the head of the STREAM head read
39015		queue is marked by some module downstream. The <i>arg</i> argument determines
39016		how the checking is done when there may be multiple marked messages on
39017		the STREAM head read queue. It may take on the following values:
39018		<code>ANYMARK</code> Check if the message is marked.
39019		<code>LASTMARK</code> Check if the message is the last one marked on the queue.
39020		The bitwise-inclusive OR of the flags <code>ANYMARK</code> and <code>LASTMARK</code> is
39021		permitted.
39022		The return value shall be 1 if the mark condition is satisfied; otherwise, the
39023		value shall be 0.
39024		The <i>ioctl()</i> function with the <code>I_ATMARK</code> command shall fail if:
39025		[EINVAL] Invalid <i>arg</i> value.
39026	<code>I_CKBAND</code>	Checks if the message of a given priority band exists on the STREAM head
39027		read queue. This shall return 1 if a message of the given priority exists, 0 if no
39028		such message exists, or <code>-1</code> on error. <i>arg</i> should be of type <code>int</code> .
39029		The <i>ioctl()</i> function with the <code>I_CKBAND</code> command shall fail if:
39030		[EINVAL] Invalid <i>arg</i> value.
39031	<code>I_GETBAND</code>	Returns the priority band of the first message on the STREAM head read
39032		queue in the integer referenced by <i>arg</i> .
39033		The <i>ioctl()</i> function with the <code>I_GETBAND</code> command shall fail if:
39034		[ENODATA] No message on the STREAM head read queue.
39035	<code>I_CANPUT</code>	Checks if a certain band is writable. <i>arg</i> is set to the priority band in question.
39036		The return value shall be 0 if the band is flow-controlled, 1 if the band is
39037		writable, or <code>-1</code> on error.
39038		The <i>ioctl()</i> function with the <code>I_CANPUT</code> command shall fail if:
39039		[EINVAL] Invalid <i>arg</i> value.
39040	<code>I_SETCLTIME</code>	This request allows the process to set the time the STREAM head shall delay
39041		when a STREAM is closing and there is data on the write queues. Before
39042		closing each module or driver, if there is data on its write queue, the STREAM
39043		head shall delay for the specified amount of time to allow the data to drain. If,
39044		after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a
39045		pointer to an integer specifying the number of milliseconds to delay, rounded
39046		up to the nearest valid value. If <code>I_SETCLTIME</code> is not performed on a STREAM,
39047		an implementation-defined default timeout interval is used.

39048 The *ioctl()* function with the I_SETCLTIME command shall fail if:

39049 [EINVAL] Invalid *arg* value.

39050 I_GETCLTIME Returns the close time delay in the integer pointed to by *arg*.

39051 **Multiplexed STREAMS Configurations**

39052 The following commands are used for connecting and disconnecting multiplexed STREAMS configurations. These commands use an implementation-defined default timeout interval.

39053

39054 I_LINK Connects two STREAMs, where *fildev* is the file descriptor of the STREAM connected to the multiplexing driver, and *arg* is the file descriptor of the STREAM connected to another driver. The STREAM designated by *arg* is connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the STREAM head regarding the connection. This call shall return a multiplexer ID number (an identifier used to disconnect the multiplexer; see I_UNLINK) on success, and -1 on failure.

39055

39056

39057

39058

39059

39060

39061

39062 The *ioctl()* function with the I_LINK command shall fail if:

39063 [ENXIO] Hangup received on *fildev*.

39064 [ETIME] Timeout before acknowledgement message was received at STREAM head.

39065

39066 [EAGAIN] or [ENOSR]

39067 Unable to allocate STREAMS storage to perform the I_LINK.

39068

39069 [EBADF] The *arg* argument is not a valid, open file descriptor.

39070 [EINVAL] The *fildev* argument does not support multiplexing; or *arg* is not a STREAM or is already connected downstream from a multiplexer; or the specified I_LINK operation would connect the STREAM head in more than one place in the multiplexed STREAM.

39071

39072

39073

39074

39075 An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of *fildev*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK fails with *errno* set to the value in the message.

39076

39077

39078

39079

39080 I_UNLINK Disconnects the two STREAMs specified by *fildev* and *arg*. *fildev* is the file descriptor of the STREAM connected to the multiplexing driver. The *arg* argument is the multiplexer ID number that was returned by the I_LINK *ioctl()* command when a STREAM was connected downstream from the multiplexing driver. If *arg* is MUXID_ALL, then all STREAMs that were connected to *fildev* shall be disconnected. As in I_LINK, this command requires acknowledgement.

39081

39082

39083

39084

39085

39086

39087 The *ioctl()* function with the I_UNLINK command shall fail if:

39088 [ENXIO] Hangup received on *fildev*.

39089		[ETIME]	Timeout before acknowledgement message was received at
39090			STREAM head.
39091		[EAGAIN] or [ENOSR]	
39092			Unable to allocate buffers for the acknowledgement
39093			message.
39094		[EINVAL]	Invalid multiplexer ID number.
39095			An I_UNLINK can also fail while waiting for the multiplexing driver to
39096			acknowledge the request if a message indicating an error or a hangup is
39097			received at the STREAM head of <i>fildev</i> . In addition, an error code can be
39098			returned in the positive or negative acknowledgement message. For these
39099			cases, I_UNLINK shall fail with <i>errno</i> set to the value in the message.
39100	I_PLINK		Creates a <i>persistent connection</i> between two STREAMs, where <i>fildev</i> is the file
39101			descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the
39102			file descriptor of the STREAM connected to another driver. This call shall
39103			create a persistent connection which can exist even if the file descriptor <i>fildev</i>
39104			associated with the upper STREAM to the multiplexing driver is closed. The
39105			STREAM designated by <i>arg</i> gets connected via a persistent connection below
39106			the multiplexing driver. I_PLINK requires the multiplexing driver to send an
39107			acknowledgement message to the STREAM head. This call shall return a
39108			multiplexer ID number (an identifier that may be used to disconnect the
39109			multiplexer; see I_PUNLINK) on success, and -1 on failure.
39110			The <i>ioctl()</i> function with the I_PLINK command shall fail if:
39111		[ENXIO]	Hangup received on <i>fildev</i> .
39112		[ETIME]	Timeout before acknowledgement message was received at
39113			STREAM head.
39114		[EAGAIN] or [ENOSR]	
39115			Unable to allocate STREAMS storage to perform the
39116			I_PLINK.
39117		[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
39118		[EINVAL]	The <i>fildev</i> argument does not support multiplexing; or <i>arg</i> is
39119			not a STREAM or is already connected downstream from a
39120			multiplexer; or the specified I_PLINK operation would
39121			connect the STREAM head in more than one place in the
39122			multiplexed STREAM.
39123			An I_PLINK can also fail while waiting for the multiplexing driver to
39124			acknowledge the request, if a message indicating an error or a hangup is
39125			received at the STREAM head of <i>fildev</i> . In addition, an error code can be
39126			returned in the positive or negative acknowledgement message. For these
39127			cases, I_PLINK shall fail with <i>errno</i> set to the value in the message.
39128	I_PUNLINK		Disconnects the two STREAMs specified by <i>fildev</i> and <i>arg</i> from a persistent
39129			connection. The <i>fildev</i> argument is the file descriptor of the STREAM
39130			connected to the multiplexing driver. The <i>arg</i> argument is the multiplexer ID
39131			number that was returned by the I_PLINK <i>ioctl()</i> command when a STREAM
39132			was connected downstream from the multiplexing driver. If <i>arg</i> is
39133			MUXID_ALL, then all STREAMs which are persistent connections to <i>fildev</i>
39134			shall be disconnected. As in I_PLINK, this command requires the multiplexing

39135 driver to acknowledge the request.

39136 The *ioctl()* function with the I_PUNLINK command shall fail if:

39137 [ENXIO] Hangup received on *fildev*.

39138 [ETIME] Timeout before acknowledgement message was received at
39139 STREAM head.

39140 [EAGAIN] or [ENOSR]
39141 Unable to allocate buffers for the acknowledgement
39142 message.

39143 [EINVAL] Invalid multiplexer ID number.

39144 An I_PUNLINK can also fail while waiting for the multiplexing driver to
39145 acknowledge the request if a message indicating an error or a hangup is
39146 received at the STREAM head of *fildev*. In addition, an error code can be
39147 returned in the positive or negative acknowledgement message. For these
39148 cases, I_PUNLINK shall fail with *errno* set to the value in the message.

39149 RETURN VALUE

39150 Upon successful completion, *ioctl()* shall return a value other than -1 that depends upon the
39151 STREAMS device control function. Otherwise, it shall return -1 and set *errno* to indicate the
39152 error.

39153 ERRORS

39154 Under the following general conditions, *ioctl()* shall fail if:

39155 [EBADF] The *fildev* argument is not a valid open file descriptor.

39156 [EINTR] A signal was caught during the *ioctl()* operation.

39157 [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or
39158 indirectly) downstream from a multiplexer.

39159 If an underlying device driver detects an error, then *ioctl()* shall fail if:

39160 [EINVAL] The *request* or *arg* argument is not valid for this device.

39161 [EIO] Some physical I/O error has occurred.

39162 [ENOTTY] The file associated with the *fildev* argument is not a STREAMS device that
39163 accepts control functions.

39164 [ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service
39165 requested cannot be performed on this particular sub-device.

39166 [ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding
39167 device driver does not support the *ioctl()* function.

39168 If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except
39169 I_UNLINK and I_PUNLINK shall set *errno* to [EINVAL].

39170 EXAMPLES

39171 None.

39172 APPLICATION USAGE

39173 The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

39174 RATIONALE

39175 None.

39176 FUTURE DIRECTIONS

39177 The *ioctl()* function may be removed in a future version.

39178 SEE ALSO

39179 [Section 2.6](#) (on page 500), *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read()*,
39180 *sigaction()*, *write()*

39181 XBD [<stropts.h>](#)

39182 CHANGE HISTORY

39183 First released in Issue 4, Version 2.

39184 Issue 5

39185 Moved from X/OPEN UNIX extension to BASE.

39186 Issue 6

39187 The Open Group Corrigendum U028/4 is applied, correcting text in the I_FDINSERT [EINVAL]
39188 case to refer to *ctlbuf*.

39189 This function is marked as part of the XSI STREAMS Option Group.

39190 The normative text is updated to avoid use of the term “must” for application requirements.

39191 Issue 7

39192 Austin Group Interpretation 1003.1-2001 #155 is applied, adding a “may fail” [EINVAL] error
39193 condition for the I_SENDFD command.

39194 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

39195 The *ioctl()* function is marked obsolescent.

39196 **NAME**

39197 isalnum, isalnum_l ‡test for an alphanumeric character

39198 **SYNOPSIS**

39199 #include <ctype.h>

39200 int isalnum(int c);

39201 CX int isalnum_l(int c, locale_t locale);

39202 **DESCRIPTION**39203 CX For *isalnum()*: The functionality described on this reference page is aligned with the ISO C
39204 standard. Any conflict between the requirements described here and the ISO C standard is
39205 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.39206 CX The *isalnum()* and *isalnum_l()* functions shall test whether *c* is a character of class **alpha** or
39207 **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7
39208 (on page 135).39209 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
39210 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
39211 behavior is undefined.39212 CX The behavior is undefined if the *locale* argument to *isalnum_l()* is the special locale object
39213 LC_GLOBAL_LOCALE or is not a valid locale object handle.39214 **RETURN VALUE**39215 CX The *isalnum()* and *isalnum_l()* functions shall return non-zero if *c* is an alphanumeric character;
39216 otherwise, they shall return 0.39217 **ERRORS**

39218 No errors are defined.

39219 **EXAMPLES**

39220 None.

39221 **APPLICATION USAGE**39222 To ensure applications portability, especially across natural languages, only these functions and
39223 the functions in the reference pages listed in the SEE ALSO section should be used for character
39224 classification.39225 **RATIONALE**

39226 None.

39227 **FUTURE DIRECTIONS**

39228 None.

39229 **SEE ALSO**39230 *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
39231 *isxdigit()*, *setlocale()*, *uselocale()*

39232 XBD Chapter 7 (on page 135), <ctype.h>, <stdio.h>

39233 **CHANGE HISTORY**

39234 First released in Issue 1. Derived from Issue 1 of the SVID.

39235 **Issue 6**

39236 The normative text is updated to avoid use of the term “must” for application requirements.

39237 **Issue 7**39238
39239

The *isalnum_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

39240
39241

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0274 [302], XSH/TC1-2008/0275 [283], and XSH/TC1-2008/0276 [283] are applied.

39242 **NAME**

39243 isalpha, isalpha_l ¶test for an alphabetic character

39244 **SYNOPSIS**

39245 #include <ctype.h>

39246 int isalpha(int c);

39247 CX int isalpha_l(int c, locale_t locale);

39248 **DESCRIPTION**39249 CX For *isalpha()*: The functionality described on this reference page is aligned with the ISO C
39250 standard. Any conflict between the requirements described here and the ISO C standard is
39251 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.39252 CX The *isalpha()* and *isalpha_l()* functions shall test whether *c* is a character of class **alpha** in the
39253 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page
39254 135).39255 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
39256 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
39257 behavior is undefined.39258 CX The behavior is undefined if the *locale* argument to *isalpha_l()* is the special locale object
39259 LC_GLOBAL_LOCALE or is not a valid locale object handle.39260 **RETURN VALUE**39261 CX The *isalpha()* and *isalpha_l()* functions shall return non-zero if *c* is an alphabetic character;
39262 otherwise, they shall return 0.39263 **ERRORS**

39264 No errors are defined.

39265 **EXAMPLES**

39266 None.

39267 **APPLICATION USAGE**39268 To ensure applications portability, especially across natural languages, only these functions and
39269 the functions in the reference pages listed in the SEE ALSO section should be used for character
39270 classification.39271 **RATIONALE**

39272 None.

39273 **FUTURE DIRECTIONS**

39274 None.

39275 **SEE ALSO**39276 *isalnum()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
39277 *isxdigit()*, *setlocale()*, *uselocale()*

39278 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>, <stdio.h>

39279 **CHANGE HISTORY**

39280 First released in Issue 1. Derived from Issue 1 of the SVID.

39281 **Issue 6**

39282 The normative text is updated to avoid use of the term “must” for application requirements.

39283 **Issue 7**

39284 The *isalpha_l()* function is added from The Open Group Technical Standard, 2006, Extended API
39285 Set Part 4.

39286 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0277 [302], XSH/TC1-2008/0278 [283],
39287 and XSH/TC1-2008/0279 [283] are applied.

39288 **NAME**

39289 isascii ‡test for a 7-bit US-ASCII character

39290 **SYNOPSIS**

```
39291 OB XSI #include <ctype.h>
39292        int isascii(int c);
```

39293 **DESCRIPTION**39294 The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.39295 The *isascii()* function is defined on all integer values.39296 **RETURN VALUE**39297 The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and
39298 octal 0177 inclusive; otherwise, it shall return 0.39299 **ERRORS**

39300 No errors are defined.

39301 **EXAMPLES**

39302 None.

39303 **APPLICATION USAGE**39304 The *isascii()* function cannot be used portably in a localized application.39305 **RATIONALE**

39306 None.

39307 **FUTURE DIRECTIONS**39308 The *isascii()* function may be removed in a future version.39309 **SEE ALSO**39310 XBD [<ctype.h>](#)39311 **CHANGE HISTORY**

39312 First released in Issue 1. Derived from Issue 1 of the SVID.

39313 **Issue 7**39314 The *isascii()* function is marked obsolescent.

39315 **NAME**39316 isastream — test a file descriptor (**STREAMS**)39317 **SYNOPSIS**

```
39318 OB XSR #include <stropts.h>  
39319 int isastream(int fildes);
```

39320 **DESCRIPTION**

39321 The *isastream()* function shall test whether *fildes*, an open file descriptor, is associated with a
39322 STREAMS-based file.

39323 **RETURN VALUE**

39324 Upon successful completion, *isastream()* shall return 1 if *fildes* refers to a STREAMS-based file
39325 and 0 if not. Otherwise, *isastream()* shall return -1 and set *errno* to indicate the error.

39326 **ERRORS**

39327 The *isastream()* function shall fail if:

39328 [EBADF] The *fildes* argument is not a valid open file descriptor.

39329 **EXAMPLES**

39330 None.

39331 **APPLICATION USAGE**

39332 None.

39333 **RATIONALE**

39334 None.

39335 **FUTURE DIRECTIONS**

39336 The *isastream()* function may be removed in a future version.

39337 **SEE ALSO**

39338 XBD [<stropts.h>](#)

39339 **CHANGE HISTORY**

39340 First released in Issue 4, Version 2.

39341 **Issue 5**

39342 Moved from X/OPEN UNIX extension to BASE.

39343 **Issue 7**

39344 The *isastream()* function is marked obsolescent.

39345 **NAME**

39346 isatty ǂ'test for a terminal device

39347 **SYNOPSIS**

39348 #include <unistd.h>

39349 int isatty(int *fildev*);39350 **DESCRIPTION**39351 The *isatty()* function shall test whether *fildev*, an open file descriptor, is associated with a
39352 terminal device.39353 **RETURN VALUE**39354 The *isatty()* function shall return 1 if *fildev* is associated with a terminal; otherwise, it shall return
39355 0 and may set *errno* to indicate the error.39356 **ERRORS**39357 The *isatty()* function may fail if:39358 [EBADF] The *fildev* argument is not a valid open file descriptor.39359 [ENOTTY] The file associated with the *fildev* argument is not a terminal.39360 **EXAMPLES**

39361 None.

39362 **APPLICATION USAGE**39363 The *isatty()* function does not necessarily indicate that a human being is available for interaction
39364 via *fildev*. It is quite possible that non-terminal devices are connected to the communications
39365 line.39366 **RATIONALE**

39367 None.

39368 **FUTURE DIRECTIONS**

39369 None.

39370 **SEE ALSO**39371 XBD <[unistd.h](#)>39372 **CHANGE HISTORY**

39373 First released in Issue 1. Derived from Issue 1 of the SVID.

39374 **Issue 6**39375 The following new requirements on POSIX implementations derive from alignment with the
39376 Single UNIX Specification:39377 The optional setting of *errno* to indicate an error is added.

39378 The [EBADF] and [ENOTTY] optional error conditions are added.

39379 **Issue 7**

39380 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

39381 **NAME**

39382 isblank, isblank_l ‡test for a blank character

39383 **SYNOPSIS**

39384 #include <ctype.h>

39385 int isblank(int c);

39386 CX int isblank_l(int c, locale_t locale);

39387 **DESCRIPTION**39388 CX For *isblank()*: The functionality described on this reference page is aligned with the ISO C
39389 standard. Any conflict between the requirements described here and the ISO C standard is
39390 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.39391 CX The *isblank()* and *isblank_l()* functions shall test whether *c* is a character of class **blank** in the
39392 CX current locale, or in the locale represented by *locale*, respectively; see XBD [Chapter 7](#) (on page
39393 135).39394 The *c* argument is a type **int**, the value of which the application shall ensure is a character
39395 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
39396 any other value, the behavior is undefined.39397 CX The behavior is undefined if the *locale* argument to *isblank_l()* is the special locale object
39398 LC_GLOBAL_LOCALE or is not a valid locale object handle.39399 **RETURN VALUE**39400 CX The *isblank()* and *isblank_l()* functions shall return non-zero if *c* is a <blank>; otherwise, they
39401 shall return 0.39402 **ERRORS**

39403 No errors are defined.

39404 **EXAMPLES**

39405 None.

39406 **APPLICATION USAGE**39407 To ensure applications portability, especially across natural languages, only these functions and
39408 the functions in the reference pages listed in the SEE ALSO section should be used for character
39409 classification.39410 **RATIONALE**

39411 None.

39412 **FUTURE DIRECTIONS**

39413 None.

39414 **SEE ALSO**39415 [isalnum\(\)](#), [isalpha\(\)](#), [iscntrl\(\)](#), [isdigit\(\)](#), [isgraph\(\)](#), [islower\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#), [isupper\(\)](#),
39416 [isxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#)39417 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)39418 **CHANGE HISTORY**

39419 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

39420 **Issue 7**39421 The *isblank_l()* function is added from The Open Group Technical Standard, 2006, Extended API
39422 Set Part 4.

39423
39424

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0280 [302], XSH/TC1-2008/0281 [283], and XSH/TC1-2008/0282 [283] are applied.

39425 **NAME**

39426 iscntrl, iscntrl_l — test for a control character

39427 **SYNOPSIS**

```
39428       #include <ctype.h>
39429       int iscntrl(int c);
39430 CX     int iscntrl_l(int c, locale_t locale);
```

39431 **DESCRIPTION**

39432 CX For *iscntrl()*: The functionality described on this reference page is aligned with the ISO C
 39433 standard. Any conflict between the requirements described here and the ISO C standard is
 39434 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

39435 CX The *iscntrl()* and *iscntrl_l()* functions shall test whether *c* is a character of class **cntrl** in the
 39436 CX current locale, or in the locale represented by *locale*, respectively; see XBD [Chapter 7](#) (on page
 39437 135).

39438 The *c* argument is a type **int**, the value of which the application shall ensure is a character
 39439 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 39440 any other value, the behavior is undefined.

39441 CX The behavior is undefined if the *locale* argument to *iscntrl_l()* is the special locale object
 39442 LC_GLOBAL_LOCALE or is not a valid locale object handle.

39443 **RETURN VALUE**

39444 CX The *iscntrl()* and *iscntrl_l()* functions shall return non-zero if *c* is a control character; otherwise,
 39445 they shall return 0.

39446 **ERRORS**

39447 No errors are defined.

39448 **EXAMPLES**

39449 None.

39450 **APPLICATION USAGE**

39451 To ensure applications portability, especially across natural languages, only these functions and
 39452 the functions in the reference pages listed in the SEE ALSO section should be used for character
 39453 classification.

39454 **RATIONALE**

39455 None.

39456 **FUTURE DIRECTIONS**

39457 None.

39458 **SEE ALSO**

39459 *isalnum()*, *isalpha()*, *isblank()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,
 39460 *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

39461 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)

39462 **CHANGE HISTORY**

39463 First released in Issue 1. Derived from Issue 1 of the SVID.

39464 **Issue 6**

39465 The normative text is updated to avoid use of the term “must” for application requirements.

39466 **Issue 7**

39467 The *iscntrl_l()* function is added from The Open Group Technical Standard, 2006, Extended API
39468 Set Part 4.

39469 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0283 [302], XSH/TC1-2008/0284 [283],
39470 and XSH/TC1-2008/0285 [283] are applied.

39471 **NAME**

39472 isdigit, isdigit_l †test for a decimal digit

39473 **SYNOPSIS**

39474 #include <ctype.h>

39475 int isdigit(int c);

39476 CX int isdigit_l(int c, locale_t locale);

39477 **DESCRIPTION**39478 CX For *isdigit()*: The functionality described on this reference page is aligned with the ISO C
39479 standard. Any conflict between the requirements described here and the ISO C standard is
39480 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.39481 CX The *isdigit()* and *isdigit_l()* functions shall test whether *c* is a character of class **digit** in the
39482 current locale, or in the locale represented by *locale*, respectively; see XBD [Chapter 7](#) (on page
39483 135).39484 The *c* argument is an **int**, the value of which the application shall ensure is a character
39485 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
39486 any other value, the behavior is undefined.39487 CX The behavior is undefined if the *locale* argument to *isdigit_l()* is the special locale object
39488 LC_GLOBAL_LOCALE or is not a valid locale object handle.39489 **RETURN VALUE**39490 CX The *isdigit()* and *isdigit_l()* functions shall return non-zero if *c* is a decimal digit; otherwise,
39491 they shall return 0.39492 **ERRORS**

39493 No errors are defined.

39494 **EXAMPLES**

39495 None.

39496 **APPLICATION USAGE**39497 To ensure applications portability, especially across natural languages, only these functions and
39498 the functions in the reference pages listed in the SEE ALSO section should be used for character
39499 classification.39500 **RATIONALE**

39501 None.

39502 **FUTURE DIRECTIONS**

39503 None.

39504 **SEE ALSO**39505 [isalnum\(\)](#), [isalpha\(\)](#), [isblank\(\)](#), [iscntrl\(\)](#), [isgraph\(\)](#), [islower\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#),
39506 [isupper\(\)](#), [isxdigit\(\)](#)39507 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)39508 **CHANGE HISTORY**

39509 First released in Issue 1. Derived from Issue 1 of the SVID.

39510 **Issue 6**

39511 The normative text is updated to avoid use of the term “must” for application requirements.

39512 **Issue 7**

39513 The *isdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API
39514 Set Part 4.

39515 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0286 [302], XSH/TC1-2008/0287 [283],
39516 and XSH/TC1-2008/0288 [283] are applied.

39517 **NAME**

39518 isfinite †'test for finite value

39519 **SYNOPSIS**

39520 #include <math.h>

39521 int isfinite(real-floating x);

39522 **DESCRIPTION**

39523 CX The functionality described on this reference page is aligned with the ISO C standard. Any
39524 conflict between the requirements described here and the ISO C standard is unintentional. This
39525 volume of POSIX.1-2017 defers to the ISO C standard.

39526 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or
39527 normal, and not infinite or NaN). First, an argument represented in a format wider than its
39528 semantic type is converted to its semantic type. Then determination is based on the type of the
39529 argument.

39530 **RETURN VALUE**39531 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.39532 **ERRORS**

39533 No errors are defined.

39534 **EXAMPLES**

39535 None.

39536 **APPLICATION USAGE**

39537 None.

39538 **RATIONALE**

39539 None.

39540 **FUTURE DIRECTIONS**

39541 None.

39542 **SEE ALSO**39543 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

39544 XBD <math.h>

39545 **CHANGE HISTORY**

39546 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

39547 **NAME**

39548 isgraph, isgraph_l ‡test for a visible character

39549 **SYNOPSIS**

39550 #include <ctype.h>

39551 int isgraph(int c);

39552 CX int isgraph_l(int c, locale_t locale);

39553 **DESCRIPTION**39554 CX For *isgraph()*: The functionality described on this reference page is aligned with the ISO C
39555 standard. Any conflict between the requirements described here and the ISO C standard is
39556 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.39557 CX The *isgraph()* and *isgraph_l()* functions shall test whether *c* is a character of class **graph** in the
39558 CX current locale, or in the locale represented by *locale*, respectively; see XBD [Chapter 7](#) (on page
39559 135).39560 The *c* argument is an **int**, the value of which the application shall ensure is a character
39561 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
39562 any other value, the behavior is undefined.39563 CX The behavior is undefined if the *locale* argument to *isgraph_l()* is the special locale object
39564 LC_GLOBAL_LOCALE or is not a valid locale object handle.39565 **RETURN VALUE**39566 CX The *isgraph()* and *isgraph_l()* functions shall return non-zero if *c* is a character with a visible
39567 representation; otherwise, they shall return 0.39568 **ERRORS**

39569 No errors are defined.

39570 **EXAMPLES**

39571 None.

39572 **APPLICATION USAGE**39573 To ensure applications portability, especially across natural languages, only these functions and
39574 the functions in the reference pages listed in the SEE ALSO section should be used for character
39575 classification.39576 **RATIONALE**

39577 None.

39578 **FUTURE DIRECTIONS**

39579 None.

39580 **SEE ALSO**39581 [isalnum\(\)](#), [isalpha\(\)](#), [isblank\(\)](#), [iscntrl\(\)](#), [isdigit\(\)](#), [islower\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#), [isupper\(\)](#),
39582 [isxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#)39583 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)39584 **CHANGE HISTORY**

39585 First released in Issue 1. Derived from Issue 1 of the SVID.

39586 **Issue 6**

39587 The normative text is updated to avoid use of the term “must” for application requirements.

39588 **Issue 7**

39589

39590

The *isgraph_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

39591

39592

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0289 [302], XSH/TC1-2008/0290 [283], and XSH/TC1-2008/0291 [283] are applied.

39593 **NAME**39594 `isgreater` — test if x greater than y 39595 **SYNOPSIS**39596 `#include <math.h>`39597 `int isgreater(real-floating x , real-floating y);`39598 **DESCRIPTION**

39599 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
39600 conflict between the requirements described here and the ISO C standard is unintentional. This
39601 volume of POSIX.1-2017 defers to the ISO C standard.

39602 The `isgreater()` macro shall determine whether its first argument is greater than its second
39603 argument. The value of `isgreater(x , y)` shall be equal to $(x) > (y)$; however, unlike $(x) > (y)$,
39604 `isgreater(x , y)` shall not raise the invalid floating-point exception when x and y are unordered.

39605 **RETURN VALUE**39606 Upon successful completion, the `isgreater()` macro shall return the value of $(x) > (y)$.39607 If x or y is NaN, 0 shall be returned.39608 **ERRORS**

39609 No errors are defined.

39610 **EXAMPLES**

39611 None.

39612 **APPLICATION USAGE**

39613 The relational and equality operators support the usual mathematical relationships between
39614 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
39615 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
39616 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
39617 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
39618 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
39619 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
39620 indicates that the argument shall be an expression of **real-floating** type.

39621 **RATIONALE**

39622 None.

39623 **FUTURE DIRECTIONS**

39624 None.

39625 **SEE ALSO**39626 [*isgreaterequal\(\)*](#), [*isless\(\)*](#), [*islessequal\(\)*](#), [*islessgreater\(\)*](#), [*isunordered\(\)*](#)39627 XBD [**<math.h>**](#)39628 **CHANGE HISTORY**

39629 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39630 **NAME**39631 `isgreaterequal` — test if x is greater than or equal to y 39632 **SYNOPSIS**39633 `#include <math.h>`39634 `int isgreaterequal(real-floating x , real-floating y);`39635 **DESCRIPTION**

39636 CX The functionality described on this reference page is aligned with the ISO C standard. Any
39637 conflict between the requirements described here and the ISO C standard is unintentional. This
39638 volume of POSIX.1-2017 defers to the ISO C standard.

39639 The `isgreaterequal()` macro shall determine whether its first argument is greater than or equal to
39640 its second argument. The value of `isgreaterequal(x , y)` shall be equal to $(x) \geq (y)$; however, unlike
39641 $(x) \geq (y)$, `isgreaterequal(x , y)` shall not raise the invalid floating-point exception when x and y are
39642 unordered.

39643 **RETURN VALUE**39644 Upon successful completion, the `isgreaterequal()` macro shall return the value of $(x) \geq (y)$.39645 If x or y is NaN, 0 shall be returned.39646 **ERRORS**

39647 No errors are defined.

39648 **EXAMPLES**

39649 None.

39650 **APPLICATION USAGE**

39651 The relational and equality operators support the usual mathematical relationships between
39652 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
39653 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
39654 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
39655 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
39656 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
39657 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
39658 indicates that the argument shall be an expression of **real-floating** type.

39659 **RATIONALE**

39660 None.

39661 **FUTURE DIRECTIONS**

39662 None.

39663 **SEE ALSO**39664 [`isgreater\(\)`](#), [`isless\(\)`](#), [`islessequal\(\)`](#), [`islessgreater\(\)`](#), [`isunordered\(\)`](#)39665 XBD [`<math.h>`](#)39666 **CHANGE HISTORY**

39667 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39668 **NAME**39669 `isinf` ‡test for infinity39670 **SYNOPSIS**39671 `#include <math.h>`39672 `int isinf(real-floating x);`39673 **DESCRIPTION**

39674 CX The functionality described on this reference page is aligned with the ISO C standard. Any
39675 conflict between the requirements described here and the ISO C standard is unintentional. This
39676 volume of POSIX.1-2017 defers to the ISO C standard.

39677 The *isinf()* macro shall determine whether its argument value is an infinity (positive or
39678 negative). First, an argument represented in a format wider than its semantic type is converted
39679 to its semantic type. Then determination is based on the type of the argument.

39680 **RETURN VALUE**39681 The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.39682 **ERRORS**

39683 No errors are defined.

39684 **EXAMPLES**

39685 None.

39686 **APPLICATION USAGE**

39687 None.

39688 **RATIONALE**

39689 None.

39690 **FUTURE DIRECTIONS**

39691 None.

39692 **SEE ALSO**39693 [*fpclassify\(\)*](#), [*isfinite\(\)*](#), [*isnan\(\)*](#), [*isnormal\(\)*](#), [*signbit\(\)*](#)39694 XBD [**<math.h>**](#)39695 **CHANGE HISTORY**

39696 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

39697 **NAME**39698 isless ‡test if x is less than y 39699 **SYNOPSIS**

39700 #include <math.h>

39701 int isless(real-floating x , real-floating y);39702 **DESCRIPTION**

39703 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 39704 conflict between the requirements described here and the ISO C standard is unintentional. This
 39705 volume of POSIX.1-2017 defers to the ISO C standard.

39706 The *isless()* macro shall determine whether its first argument is less than its second argument.
 39707 The value of *isless*(x , y) shall be equal to $(x) < (y)$; however, unlike $(x) < (y)$, *isless*(x , y) shall not
 39708 raise the invalid floating-point exception when x and y are unordered.

39709 **RETURN VALUE**39710 Upon successful completion, the *isless()* macro shall return the value of $(x) < (y)$.39711 If x or y is NaN, 0 shall be returned.39712 **ERRORS**

39713 No errors are defined.

39714 **EXAMPLES**

39715 None.

39716 **APPLICATION USAGE**

39717 The relational and equality operators support the usual mathematical relationships between
 39718 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 39719 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 39720 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 39721 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 39722 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 39723 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 39724 indicates that the argument shall be an expression of **real-floating** type.

39725 **RATIONALE**

39726 None.

39727 **FUTURE DIRECTIONS**

39728 None.

39729 **SEE ALSO**39730 *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*

39731 XBD <math.h>

39732 **CHANGE HISTORY**

39733 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39734 **NAME**39735 islessequal ⚔test if x is less than or equal to y 39736 **SYNOPSIS**

39737 #include <math.h>

39738 int islessequal(real-floating x , real-floating y);39739 **DESCRIPTION**

39740 CX The functionality described on this reference page is aligned with the ISO C standard. Any
39741 conflict between the requirements described here and the ISO C standard is unintentional. This
39742 volume of POSIX.1-2017 defers to the ISO C standard.

39743 The *islessequal()* macro shall determine whether its first argument is less than or equal to its
39744 second argument. The value of *islessequal*(x , y) shall be equal to $(x) \leq (y)$; however, unlike
39745 $(x) \leq (y)$, *islessequal*(x , y) shall not raise the invalid floating-point exception when x and y are
39746 unordered.

39747 **RETURN VALUE**39748 Upon successful completion, the *islessequal()* macro shall return the value of $(x) \leq (y)$.39749 If x or y is NaN, 0 shall be returned.39750 **ERRORS**

39751 No errors are defined.

39752 **EXAMPLES**

39753 None.

39754 **APPLICATION USAGE**

39755 The relational and equality operators support the usual mathematical relationships between
39756 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
39757 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
39758 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
39759 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
39760 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
39761 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
39762 indicates that the argument shall be an expression of **real-floating** type.

39763 **RATIONALE**

39764 None.

39765 **FUTURE DIRECTIONS**

39766 None.

39767 **SEE ALSO**39768 *isgreater()*, *isgreaterequal()*, *isless()*, *islessgreater()*, *isunordered()*

39769 XBD <math.h>

39770 **CHANGE HISTORY**

39771 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39772 **NAME**39773 `islessgreater` — test if x is less than or greater than y 39774 **SYNOPSIS**39775 `#include <math.h>`39776 `int islessgreater(real-floating x , real-floating y);`39777 **DESCRIPTION**

39778 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39779 conflict between the requirements described here and the ISO C standard is unintentional. This
 39780 volume of POSIX.1-2017 defers to the ISO C standard.

39781 The `islessgreater()` macro shall determine whether its first argument is less than or greater than
 39782 its second argument. The `islessgreater(x , y)` macro is similar to $(x) < (y) \parallel (x) > (y)$; however,
 39783 `islessgreater(x , y)` shall not raise the invalid floating-point exception when x and y are unordered
 39784 (nor shall it evaluate x and y twice).

39785 **RETURN VALUE**

39786 Upon successful completion, the `islessgreater()` macro shall return the value of
 39787 $(x) < (y) \parallel (x) > (y)$.

39788 If x or y is NaN, 0 shall be returned.

39789 **ERRORS**

39790 No errors are defined.

39791 **EXAMPLES**

39792 None.

39793 **APPLICATION USAGE**

39794 The relational and equality operators support the usual mathematical relationships between
 39795 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 39796 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 39797 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 39798 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 39799 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 39800 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 39801 indicates that the argument shall be an expression of **real-floating** type.

39802 **RATIONALE**

39803 None.

39804 **FUTURE DIRECTIONS**

39805 None.

39806 **SEE ALSO**

39807 [*isgreater\(\)*](#), [*isgreaterequal\(\)*](#), [*isless\(\)*](#), [*islessequal\(\)*](#), [*isunordered\(\)*](#)

39808 XBD [**<math.h>**](#)

39809 **CHANGE HISTORY**

39810 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39811 **NAME**

39812 islower, islower_l — test for a lowercase letter

39813 **SYNOPSIS**

```
39814 #include <ctype.h>
39815 int islower(int c);
39816 CX int islower_l(int c, locale_t locale);
```

39817 **DESCRIPTION**

39818 CX For *islower()*: The functionality described on this reference page is aligned with the ISO C
 39819 standard. Any conflict between the requirements described here and the ISO C standard is
 39820 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

39821 CX The *islower()* and *islower_l()* functions shall test whether *c* is a character of class **lower** in the
 39822 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page
 39823 135).

39824 The *c* argument is an **int**, the value of which the application shall ensure is a character
 39825 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 39826 any other value, the behavior is undefined.

39827 CX The behavior is undefined if the *locale* argument to *islower_l()* is the special locale object
 39828 LC_GLOBAL_LOCALE or is not a valid locale object handle.

39829 **RETURN VALUE**

39830 CX The *islower()* and *islower_l()* functions shall return non-zero if *c* is a lowercase letter; otherwise,
 39831 they shall return 0.

39832 **ERRORS**

39833 No errors are defined.

39834 **EXAMPLES**39835 **Testing for a Lowercase Letter**

39836 Two examples follow, the first using *islower()*, the second using multiple concurrent locales and
 39837 *islower_l()*.

39838 The examples test whether the value is a lowercase letter, based on the current locale, then use it
 39839 as part of a key value.

```
39840 /* Example 1 -- using islower() */
39841 #include <ctype.h>
39842 #include <stdlib.h>
39843 #include <locale.h>
39844 ...
39845 char *keyst;
39846 int elementlen, len;
39847 unsigned char c;
39848 ...
39849 setlocale(LC_ALL, "");
39850 ...
39851 len = 0;
39852 while (len < elementlen) {
39853     c = (unsigned char) (rand() % 256);
39854     ...
```

```

39855         if (islower(c))
39856             keystr[len++] = c;
39857         }
39858     ...
39859     /* Example 2 -- using islower_l() */
39860     #include <ctype.h>
39861     #include <stdlib.h>
39862     #include <locale.h>
39863     ...
39864     char *keystr;
39865     int elementlen, len;
39866     unsigned char c;
39867     ...
39868     locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
39869     ...
39870     len = 0;
39871     while (len < elementlen) {
39872         c = (unsigned char) (rand() % 256);
39873         ...
39874         if (islower_l(c, loc))
39875             keystr[len++] = c;
39876     }
39877     ...

```

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 135), [<ctype.h>](#), [<locale.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements and an example is added.

Issue 7

The *islower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

39898
39899
39900

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0292 [302], XSH/TC1-2008/0293 [283], XSH/TC1-2008/0294 [283], XSH/TC1-2008/0295 [302], and XSH/TC1-2008/0296 [304] are applied.

39901 **NAME**

39902 isnan ⚭test for a NaN

39903 **SYNOPSIS**

39904 #include <math.h>

39905 int isnan(real-floating x);

39906 **DESCRIPTION**

39907 CX The functionality described on this reference page is aligned with the ISO C standard. Any
39908 conflict between the requirements described here and the ISO C standard is unintentional. This
39909 volume of POSIX.1-2017 defers to the ISO C standard.

39910 The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument
39911 represented in a format wider than its semantic type is converted to its semantic type. Then
39912 determination is based on the type of the argument.

39913 **RETURN VALUE**39914 The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.39915 **ERRORS**

39916 No errors are defined.

39917 **EXAMPLES**

39918 None.

39919 **APPLICATION USAGE**

39920 None.

39921 **RATIONALE**

39922 None.

39923 **FUTURE DIRECTIONS**

39924 None.

39925 **SEE ALSO**39926 [fpclassify\(\)](#), [isfinite\(\)](#), [isinf\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)39927 XBD [<math.h>](#)39928 **CHANGE HISTORY**

39929 First released in Issue 3.

39930 **Issue 5**

39931 The DESCRIPTION is updated to indicate the return value when NaN is not supported. This
39932 text was previously published in the APPLICATION USAGE section.

39933 **Issue 6**

39934 Re-written for alignment with the ISO/IEC 9899:1999 standard.

39935 **NAME**

39936 isnormal ‡'test for a normal value

39937 **SYNOPSIS**

39938 #include <math.h>

39939 int isnormal(real-floating x);

39940 **DESCRIPTION**

39941 CX The functionality described on this reference page is aligned with the ISO C standard. Any
39942 conflict between the requirements described here and the ISO C standard is unintentional. This
39943 volume of POSIX.1-2017 defers to the ISO C standard.

39944 The *isnormal()* macro shall determine whether its argument value is normal (neither zero,
39945 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its
39946 semantic type is converted to its semantic type. Then determination is based on the type of the
39947 argument.

39948 **RETURN VALUE**

39949 The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal
39950 value.

39951 **ERRORS**

39952 No errors are defined.

39953 **EXAMPLES**

39954 None.

39955 **APPLICATION USAGE**

39956 None.

39957 **RATIONALE**

39958 None.

39959 **FUTURE DIRECTIONS**

39960 None.

39961 **SEE ALSO**39962 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*

39963 XBD <math.h>

39964 **CHANGE HISTORY**

39965 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39966 **NAME**

39967 isprint, isprint_l ‡test for a printable character

39968 **SYNOPSIS**

39969 #include <ctype.h>

39970 int isprint(int c);

39971 CX int isprint_l(int c, locale_t locale);

39972 **DESCRIPTION**39973 CX For *isprint()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.39976 CX The *isprint()* and *isprint_l()* functions shall test whether *c* is a character of class **print** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).39979 The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.39982 CX The behavior is undefined if the *locale* argument to *isprint_l()* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.39984 **RETURN VALUE**39985 CX The *isprint()* and *isprint_l()* functions shall return non-zero if *c* is a printable character; otherwise, they shall return 0.39987 **ERRORS**

39988 No errors are defined.

39989 **EXAMPLES**

39990 None.

39991 **APPLICATION USAGE**

39992 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

39995 **RATIONALE**

39996 None.

39997 **FUTURE DIRECTIONS**

39998 None.

39999 **SEE ALSO**40000 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

40002 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

40003 **CHANGE HISTORY**

40004 First released in Issue 1. Derived from Issue 1 of the SVID.

40005 **Issue 6**

40006 The normative text is updated to avoid use of the term “must” for application requirements.

40007 **Issue 7**

40008 The *isprint_l()* function is added from The Open Group Technical Standard, 2006, Extended API
40009 Set Part 4.

40010 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0297 [302], XSH/TC1-2008/0298 [283],
40011 and XSH/TC1-2008/0299 [283] are applied.

40012 **NAME**

40013 ispunct, ispunct_l ‡test for a punctuation character

40014 **SYNOPSIS**

40015 #include <ctype.h>

40016 int ispunct(int c);

40017 CX int ispunct_l(int c, locale_t locale);

40018 **DESCRIPTION**40019 CX For *ispunct()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40022 CX The *ispunct()* and *ispunct_l()* functions shall test whether *c* is a character of class **punct** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).40025 The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.40028 CX The behavior is undefined if the *locale* argument to *ispunct_l()* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.40030 **RETURN VALUE**40031 CX The *ispunct()* and *ispunct_l()* functions shall return non-zero if *c* is a punctuation character; otherwise, they shall return 0.40033 **ERRORS**

40034 No errors are defined.

40035 **EXAMPLES**

40036 None.

40037 **APPLICATION USAGE**

40038 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40041 **RATIONALE**

40042 None.

40043 **FUTURE DIRECTIONS**

40044 None.

40045 **SEE ALSO**40046 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

40048 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

40049 **CHANGE HISTORY**

40050 First released in Issue 1. Derived from Issue 1 of the SVID.

40051 **Issue 6**

40052 The normative text is updated to avoid use of the term “must” for application requirements.

40053 **Issue 7**

40054 The *ispunct_l()* function is added from The Open Group Technical Standard, 2006, Extended API
40055 Set Part 4.

40056 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0300 [302], XSH/TC1-2008/0301 [283],
40057 and XSH/TC1-2008/0302 [283] are applied.

40058 **NAME**

40059 isspace, isspace_l †'test for a white-space character

40060 **SYNOPSIS**

40061 #include <ctype.h>

40062 int isspace(int c);

40063 CX int isspace_l(int c, locale_t locale);

40064 **DESCRIPTION**40065 CX For *isspace()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40068 CX The *isspace()* and *isspace_l()* functions shall test whether *c* is a character of class **space** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).40071 The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.40074 CX The behavior is undefined if the *locale* argument to *isspace_l()* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.40076 **RETURN VALUE**40077 CX The *isspace()* and *isspace_l()* functions shall return non-zero if *c* is a white-space character; otherwise, they shall return 0.40079 **ERRORS**

40080 No errors are defined.

40081 **EXAMPLES**

40082 None.

40083 **APPLICATION USAGE**

40084 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40087 **RATIONALE**

40088 None.

40089 **FUTURE DIRECTIONS**

40090 None.

40091 **SEE ALSO**40092 *isalnum()*, *isalpha()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

40094 XBD Chapter 7 (on page 135), <ctype.h>, <locale.h>

40095 **CHANGE HISTORY**

40096 First released in Issue 1. Derived from Issue 1 of the SVID.

40097 **Issue 6**

40098 The normative text is updated to avoid use of the term ``must'' for application requirements.

40099 **Issue 7**

40100 The *isspace_l()* function is added from The Open Group Technical Standard, 2006, Extended API
40101 Set Part 4.

40102 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0303 [302], XSH/TC1-2008/0304 [283],
40103 and XSH/TC1-2008/0305 [283] are applied.

40104 **NAME**

40105 isunordered — test if arguments are unordered

40106 **SYNOPSIS**

40107 #include <math.h>

40108 int isunordered(real-floating *x*, real-floating *y*);40109 **DESCRIPTION**

40110 CX The functionality described on this reference page is aligned with the ISO C standard. Any
40111 conflict between the requirements described here and the ISO C standard is unintentional. This
40112 volume of POSIX.1-2017 defers to the ISO C standard.

40113 The *isunordered()* macro shall determine whether its arguments are unordered.40114 **RETURN VALUE**40115 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are
40116 unordered, and 0 otherwise.40117 If *x* or *y* is NaN, 1 shall be returned.40118 **ERRORS**

40119 No errors are defined.

40120 **EXAMPLES**

40121 None.

40122 **APPLICATION USAGE**

40123 The relational and equality operators support the usual mathematical relationships between
40124 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
40125 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
40126 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
40127 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
40128 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
40129 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
40130 indicates that the argument shall be an expression of **real-floating** type.

40131 **RATIONALE**

40132 None.

40133 **FUTURE DIRECTIONS**

40134 None.

40135 **SEE ALSO**40136 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*

40137 XBD <math.h>

40138 **CHANGE HISTORY**

40139 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

40140 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/50 is applied, correcting the RETURN
40141 VALUE section when *x* or *y* is NaN.

40142 **NAME**

40143 isupper, isupper_l — test for an uppercase letter

40144 **SYNOPSIS**

```
40145 #include <ctype.h>
40146 int isupper(int c);
40147 CX int isupper_l(int c, locale_t locale);
```

40148 **DESCRIPTION**

40149 CX For *isupper()*: The functionality described on this reference page is aligned with the ISO C
 40150 standard. Any conflict between the requirements described here and the ISO C standard is
 40151 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

40152 CX The *isupper()* and *isupper_l()* functions shall test whether *c* is a character of class **upper** in the
 40153 CX current locale, or in the locale represented by *locale*, respectively; see XBD [Chapter 7](#) (on page
 40154 135).

40155 The *c* argument is an **int**, the value of which the application shall ensure is a character
 40156 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 40157 any other value, the behavior is undefined.

40158 CX The behavior is undefined if the *locale* argument to *isupper_l()* is the special locale object
 40159 LC_GLOBAL_LOCALE or is not a valid locale object handle.

40160 **RETURN VALUE**

40161 CX The *isupper()* and *isupper_l()* functions shall return non-zero if *c* is an uppercase letter;
 40162 otherwise, they shall return 0.

40163 **ERRORS**

40164 No errors are defined.

40165 **EXAMPLES**

40166 None.

40167 **APPLICATION USAGE**

40168 To ensure applications portability, especially across natural languages, only these functions and
 40169 the functions in the reference pages listed in the SEE ALSO section should be used for character
 40170 classification.

40171 **RATIONALE**

40172 None.

40173 **FUTURE DIRECTIONS**

40174 None.

40175 **SEE ALSO**

40176 [isalnum\(\)](#), [isalpha\(\)](#), [isblank\(\)](#), [iscntrl\(\)](#), [isdigit\(\)](#), [isgraph\(\)](#), [islower\(\)](#), [isprint\(\)](#), [ispunct\(\)](#), [isspace\(\)](#),
 40177 [isxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#)

40178 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)40179 **CHANGE HISTORY**

40180 First released in Issue 1. Derived from Issue 1 of the SVID.

40181 **Issue 6**

40182 The normative text is updated to avoid use of the term “must” for application requirements.

40183 **Issue 7**

40184 The *isupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API
40185 Set Part 4.

40186 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0306 [302], XSH/TC1-2008/0307 [283],
40187 and XSH/TC1-2008/0308 [283] are applied.

40188 **NAME**

40189 iswalnum, iswalnum_l ‡test for an alphanumeric wide-character code

40190 **SYNOPSIS**

40191 #include <wctype.h>

40192 int iswalnum(wint_t wc);

40193 CX int iswalnum_l(wint_t wc, locale_t locale);

40194 **DESCRIPTION**40195 CX For *iswalnum()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40198 CX The *iswalnum()* and *iswalnum_l()* functions shall test whether *wc* is a wide-character code representing a character of class **alpha** or **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).40201 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.40204 CX The behavior is undefined if the *locale* argument to *iswalnum_l()* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.40206 **RETURN VALUE**40207 CX The *iswalnum()* and *iswalnum_l()* functions shall return non-zero if *wc* is an alphanumeric wide-character code; otherwise, they shall return 0.40209 **ERRORS**

40210 No errors are defined.

40211 **EXAMPLES**

40212 None.

40213 **APPLICATION USAGE**

40214 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40217 **RATIONALE**

40218 None.

40219 **FUTURE DIRECTIONS**

40220 None.

40221 **SEE ALSO**40222 *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40224 XBD Chapter 7 (on page 135), <locale.h>, <stdio.h>, <wctype.h>

40225 **CHANGE HISTORY**

40226 First released as a World-wide Portability Interface in Issue 4.

40227 **Issue 5**

40228 The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

40230 The SYNOPSIS has been changed to indicate that this function and associated data types
40231 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40232 **Issue 6**

40233 The normative text is updated to avoid use of the term “must” for application requirements.

40234 **Issue 7**

40235 The `iswalnum_l()` function is added from The Open Group Technical Standard, 2006, Extended
40236 API Set Part 4.

40237 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0309 [302], XSH/TC1-2008/0310 [283],
40238 and XSH/TC1-2008/0311 [283] are applied.

40239 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0180 [685] is applied.

40240 **NAME**

40241 iswalpha, iswalpha_l ¶test for an alphabetic wide-character code

40242 **SYNOPSIS**

40243 #include <wctype.h>

40244 int iswalpha(wint_t wc);

40245 CX int iswalpha_l(wint_t wc, locale_t locale);

40246 **DESCRIPTION**40247 CX For *iswalpha()*: The functionality described on this reference page is aligned with the ISO C
40248 standard. Any conflict between the requirements described here and the ISO C standard is
40249 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40250 CX The *iswalpha()* and *iswalpha_l()* functions shall test whether *wc* is a wide-character code
40251 CX representing a character of class **alpha** in the current locale, or in the locale represented by *locale*,
40252 respectively; see XBD Chapter 7 (on page 135).40253 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
40254 code corresponding to a valid character in the locale used by the function, or equal to the value
40255 of the macro WEOF. If the argument has any other value, the behavior is undefined.40256 CX The behavior is undefined if the *locale* argument to *iswalpha_l()* is the special locale object
40257 LC_GLOBAL_LOCALE or is not a valid locale object handle.40258 **RETURN VALUE**40259 CX The *iswalpha()* and *iswalpha_l()* functions shall return non-zero if *wc* is an alphabetic wide-
40260 character code; otherwise, they shall return 0.40261 **ERRORS**

40262 No errors are defined.

40263 **EXAMPLES**

40264 None.

40265 **APPLICATION USAGE**40266 To ensure applications portability, especially across natural languages, only these functions and
40267 the functions in the reference pages listed in the SEE ALSO section should be used for character
40268 classification.40269 **RATIONALE**

40270 None.

40271 **FUTURE DIRECTIONS**

40272 None.

40273 **SEE ALSO**40274 *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
40275 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40276 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40277 **CHANGE HISTORY**

40278 First released in Issue 4.

40279 **Issue 5**40280 The following change has been made in this version for alignment with
40281 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40282 The SYNOPSIS has been changed to indicate that this function and associated data types
40283 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40284 **Issue 6**

40285 The normative text is updated to avoid use of the term “must” for application requirements.

40286 **Issue 7**

40287 The `iswalpha_l()` function is added from The Open Group Technical Standard, 2006, Extended
40288 API Set Part 4.

40289 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0312 [302], XSH/TC1-2008/0313 [283],
40290 and XSH/TC1-2008/0314 [283] are applied.

40291 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0181 [685] is applied.

40292 **NAME**

40293 iswblank, iswblank_l ‡test for a blank wide-character code

40294 **SYNOPSIS**

40295 #include <wctype.h>

40296 int iswblank(wint_t wc);

40297 CX int iswblank_l(wint_t wc, locale_t locale);

40298 **DESCRIPTION**40299 CX For *iswblank()*: The functionality described on this reference page is aligned with the ISO C
40300 standard. Any conflict between the requirements described here and the ISO C standard is
40301 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40302 CX The *iswblank()* and *iswblank_l()* functions shall test whether *wc* is a wide-character code
40303 CX representing a character of class **blank** in the current locale, or in the locale represented by
40304 *locale*, respectively; see XBD Chapter 7 (on page 135).40305 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
40306 code corresponding to a valid character in the locale used by the function, or equal to the value
40307 of the macro WEOF. If the argument has any other value, the behavior is undefined.40308 CX The behavior is undefined if the *locale* argument to *iswblank_l()* is the special locale object
40309 LC_GLOBAL_LOCALE or is not a valid locale object handle.40310 **RETURN VALUE**40311 CX The *iswblank()* and *iswblank_l()* functions shall return non-zero if *wc* is a blank wide-character
40312 code; otherwise, they shall return 0.40313 **ERRORS**

40314 No errors are defined.

40315 **EXAMPLES**

40316 None.

40317 **APPLICATION USAGE**40318 To ensure applications portability, especially across natural languages, only these functions and
40319 the functions in the reference pages listed in the SEE ALSO section should be used for character
40320 classification.40321 **RATIONALE**

40322 None.

40323 **FUTURE DIRECTIONS**

40324 None.

40325 **SEE ALSO**40326 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
40327 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40328 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40329 **CHANGE HISTORY**

40330 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

40331 **Issue 7**40332 The *iswblank_l()* function is added from The Open Group Technical Standard, 2006, Extended
40333 API Set Part 4.

40334
40335
40336

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0315 [302], XSH/TC1-2008/0316 [283], and XSH/TC1-2008/0317 [283] are applied.
POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0182 [685] is applied.

40337 **NAME**

40338 iswcntrl, iswcntrl_l — test for a control wide-character code

40339 **SYNOPSIS**

40340 #include <wctype.h>

40341 int iswcntrl(wint_t wc);

40342 CX int iswcntrl_l(wint_t wc, locale_t locale);

40343 **DESCRIPTION**40344 CX For *iswcntrl()*: The functionality described on this reference page is aligned with the ISO C
40345 standard. Any conflict between the requirements described here and the ISO C standard is
40346 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40347 CX The *iswcntrl()* and *iswcntrl_l()* functions shall test whether *wc* is a wide-character code
40348 CX representing a character of class **cntrl** in the current locale, or in the locale represented by *locale*,
40349 respectively; see XBD Chapter 7 (on page 135).40350 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
40351 code corresponding to a valid character in the locale used by the function, or equal to the value
40352 of the macro WEOF. If the argument has any other value, the behavior is undefined.40353 CX The behavior is undefined if the *locale* argument to *iswcntrl_l()* is the special locale object
40354 LC_GLOBAL_LOCALE or is not a valid locale object handle.40355 **RETURN VALUE**40356 CX The *iswcntrl()* and *iswcntrl_l()* functions shall return non-zero if *wc* is a control wide-character
40357 code; otherwise, they shall return 0.40358 **ERRORS**

40359 No errors are defined.

40360 **EXAMPLES**

40361 None.

40362 **APPLICATION USAGE**40363 To ensure applications portability, especially across natural languages, only these functions and
40364 the functions in the reference pages listed in the SEE ALSO section should be used for character
40365 classification.40366 **RATIONALE**

40367 None.

40368 **FUTURE DIRECTIONS**

40369 None.

40370 **SEE ALSO**40371 *iswalnum()*, *iswalpha()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
40372 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40373 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40374 **CHANGE HISTORY**

40375 First released in Issue 4.

40376 **Issue 5**40377 The following change has been made in this version for alignment with
40378 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40379 The SYNOPSIS has been changed to indicate that this function and associated data types
40380 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40381 **Issue 6**

40382 The normative text is updated to avoid use of the term “must” for application requirements.

40383 **Issue 7**

40384 The `iswcntrl_l()` function is added from The Open Group Technical Standard, 2006, Extended
40385 API Set Part 4.

40386 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0318 [302], XSH/TC1-2008/0319 [283],
40387 and XSH/TC1-2008/0320 [283] are applied.

40388 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0183 [685] is applied.

40389 **NAME**40390 `iswctype, iswctype_l` ‡test character for a specified class40391 **SYNOPSIS**

```
40392     #include <wctype.h>
40393     int iswctype(wint_t wc, wctype_t charclass);
40394 CX    int iswctype_l(wint_t wc, wctype_t charclass,
40395                    locale_t locale);
```

40396 **DESCRIPTION**

40397 CX For `iswctype()`: The functionality described on this reference page is aligned with the ISO C
 40398 standard. Any conflict between the requirements described here and the ISO C standard is
 40399 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

40400 CX The `iswctype()` and `iswctype_l()` functions shall determine whether the wide-character code `wc`
 40401 CX has the character class `charclass`, returning true or false. The `iswctype()` and `iswctype_l()`
 40402 functions are defined on WEOF and wide-character codes corresponding to the valid character
 40403 CX encodings in the current locale, or in the locale represented by `locale`, respectively. If the `wc`
 40404 argument is not in the domain of the function, the result is undefined. If the value of `charclass` is
 40405 invalid (that is, not obtained by a call to `wctype()` or `charclass` is invalidated by a subsequent call
 40406 to `setlocale()` that has affected category `LC_CTYPE`) the result is unspecified.

40407 CX The behavior is undefined if the `locale` argument to `iswctype_l()` is the special locale object
 40408 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

40409 **RETURN VALUE**

40410 CX The `iswctype()` and `iswctype_l()` functions shall return non-zero (true) if and only if `wc` has the
 40411 CX property described by `charclass`. If `charclass` is `(wctype_t)0`, these functions shall return 0.

40412 **ERRORS**

40413 No errors are defined.

40414 **EXAMPLES**40415 **Testing for a Valid Character**

```
40416     #include <wctype.h>
40417     ...
40418     int yes_or_no;
40419     wint_t wc;
40420     wctype_t valid_class;
40421     ...
40422     if ((valid_class=wctype("vowel")) == (wctype_t)0)
40423         /* Invalid character class. */
40424     yes_or_no=iswctype(wc,valid_class);
```

40425 **APPLICATION USAGE**

40426 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",
 40427 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard
 40428 character classes. In the table below, the functions in the left column are equivalent to the
 40429 functions in the right column.

40430	<code>iswalnum(wc)</code>	<code>iswctype(wc, wctype("alnum"))</code>
40431	<code>iswalnum_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alnum"), locale)</code>
40432	<code>iswalpha(wc)</code>	<code>iswctype(wc, wctype("alpha"))</code>
40433	<code>iswalpha_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alpha"), locale)</code>

```

40434     iswblank(wc)                iswctype(wc, wctype("blank"))
40435     iswblank_l(wc, locale)      iswctype_l(wc, wctype("blank"), locale)
40436     iswcntrl(wc)               iswctype(wc, wctype("cntrl"))
40437     iswcntrl_l(wc, locale)     iswctype_l(wc, wctype("cntrl"), locale)
40438     iswdigit(wc)               iswctype(wc, wctype("digit"))
40439     iswdigit_l(wc, locale)     iswctype_l(wc, wctype("digit"), locale)
40440     iswgraph(wc)               iswctype(wc, wctype("graph"))
40441     iswgraph_l(wc, locale)     iswctype_l(wc, wctype("graph"), locale)
40442     iswlower(wc)               iswctype(wc, wctype("lower"))
40443     iswlower_l(wc, locale)     iswctype_l(wc, wctype("lower"), locale)
40444     iswprint(wc)               iswctype(wc, wctype("print"))
40445     iswprint_l(wc, locale)     iswctype_l(wc, wctype("print"), locale)
40446     iswpunct(wc)               iswctype(wc, wctype("punct"))
40447     iswpunct_l(wc, locale)     iswctype_l(wc, wctype("punct"), locale)
40448     iswspace(wc)               iswctype(wc, wctype("space"))
40449     iswspace_l(wc, locale)     iswctype_l(wc, wctype("space"), locale)
40450     iswupper(wc)               iswctype(wc, wctype("upper"))
40451     iswupper_l(wc, locale)     iswctype_l(wc, wctype("upper"), locale)
40452     iswxdigit(wc)              iswctype(wc, wctype("xdigit"))
40453     iswxdigit_l(wc, locale)    iswctype_l(wc, wctype("xdigit"), locale)

```

40454 RATIONALE

40455 None.

40456 FUTURE DIRECTIONS

40457 None.

40458 SEE ALSO

40459 [iswalnum\(\)](#), [iswalpha\(\)](#), [iswcntrl\(\)](#), [iswdigit\(\)](#), [iswgraph\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#),
40460 [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#), [wctype\(\)](#)

40461 XBD [<locale.h>](#), [<wctype.h>](#)

40462 CHANGE HISTORY

40463 First released as World-wide Portability Interfaces in Issue 4.

40464 Issue 5

40465 The following change has been made in this version for alignment with
40466 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40467 The SYNOPSIS has been changed to indicate that this function and associated data types
40468 are now made visible by inclusion of the [<wctype.h>](#) header rather than [<wchar.h>](#).

40469 Issue 6

40470 The behavior of `charclass = (wctype_t)0` is now described.

40471 An example is added.

40472 A new function, [iswblank\(\)](#), is added to the list in the APPLICATION USAGE.

40473 Issue 7

40474 The [iswctype_l\(\)](#) function is added from The Open Group Technical Standard, 2006, Extended
40475 API Set Part 4.

40476 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0321 [283] and XSH/TC1-2008/0322
40477 [283] are applied.

40478
40479

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0184 [799] and XSH/TC2-2008/0185 [799] are applied.

40480 **NAME**

40481 iswdigit, iswdigit_l ‡test for a decimal digit wide-character code

40482 **SYNOPSIS**

40483 #include <wctype.h>

40484 int iswdigit(wint_t wc);

40485 CX int iswdigit_l(wint_t wc, locale_t locale);

40486 **DESCRIPTION**40487 CX For *iswdigit()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40490 CX The *iswdigit()* and *iswdigit_l()* functions shall test whether *wc* is a wide-character code representing a character of class **digit** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).40493 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.40496 CX The behavior is undefined if the *locale* argument to *iswdigit_l()* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.40498 **RETURN VALUE**40499 CX The *iswdigit()* and *iswdigit_l()* functions shall return non-zero if *wc* is a decimal digit wide-character code; otherwise, they shall return 0.40501 **ERRORS**

40502 No errors are defined.

40503 **EXAMPLES**

40504 None.

40505 **APPLICATION USAGE**

40506 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40509 **RATIONALE**

40510 None.

40511 **FUTURE DIRECTIONS**

40512 None.

40513 **SEE ALSO**40514 *iswalnum()*, *iswalphabeta()*, *iswcntrl()*, *iswctype()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40516 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40517 **CHANGE HISTORY**

40518 First released in Issue 4.

40519 **Issue 5**40520 The following change has been made in this version for alignment with
40521 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40522 The SYNOPSIS has been changed to indicate that this function and associated data types
40523 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40524 **Issue 6**

40525 The normative text is updated to avoid use of the term “must” for application requirements.

40526 **Issue 7**

40527 The `iswdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended
40528 API Set Part 4.

40529 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0323 [302], XSH/TC1-2008/0324 [283],
40530 and XSH/TC1-2008/0325 [283] are applied.

40531 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0186 [685] is applied.

40532 **NAME**40533 `iswgraph, iswgraph_l` ‡test for a visible wide-character code40534 **SYNOPSIS**40535 `#include <wctype.h>`40536 `int iswgraph(wint_t wc);`40537 CX `int iswgraph_l(wint_t wc, locale_t locale);`40538 **DESCRIPTION**40539 CX For `iswgraph()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40542 CX The `iswgraph()` and `iswgraph_l()` functions shall test whether `wc` is a wide-character code representing a character of class **graph** in the current locale, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).40545 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.40548 CX The behavior is undefined if the `locale` argument to `iswgraph_l()` is the special locale object `LC_GLOBAL_LOCALE` or is not a valid locale object handle.40550 **RETURN VALUE**40551 CX The `iswgraph()` and `iswgraph_l()` functions shall return non-zero if `wc` is a wide-character code with a visible representation; otherwise, they shall return 0.40553 **ERRORS**

40554 No errors are defined.

40555 **EXAMPLES**

40556 None.

40557 **APPLICATION USAGE**

40558 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40561 **RATIONALE**

40562 None.

40563 **FUTURE DIRECTIONS**

40564 None.

40565 **SEE ALSO**40566 [iswalnum\(\)](#), [iswalphalpha\(\)](#), [iswcntrl\(\)](#), [iswctype\(\)](#), [iswdigit\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#), [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#), [setlocale\(\)](#), [uselocale\(\)](#)40568 XBD Chapter 7 (on page 135), [<locale.h>](#), [<wctype.h>](#)40569 **CHANGE HISTORY**

40570 First released in Issue 4.

40571 **Issue 5**40572 The following change has been made in this version for alignment with
40573 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40574 The SYNOPSIS has been changed to indicate that this function and associated data types
40575 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40576 **Issue 6**

40577 The normative text is updated to avoid use of the term “must” for application requirements.

40578 **Issue 7**

40579 The `iswgraph_l()` function is added from The Open Group Technical Standard, 2006, Extended
40580 API Set Part 4.

40581 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0326 [302], XSH/TC1-2008/0327 [283],
40582 and XSH/TC1-2008/0328 [283] are applied.

40583 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0187 [685] is applied.

40584 **NAME**

40585 iswlower, iswlower_l — test for a lowercase letter wide-character code

40586 **SYNOPSIS**

```
40587 #include <wctype.h>
40588 int iswlower(wint_t wc);
40589 CX int iswlower_l(wint_t wc, locale_t locale);
```

40590 **DESCRIPTION**

40591 CX For *iswlower()*: The functionality described on this reference page is aligned with the ISO C
 40592 standard. Any conflict between the requirements described here and the ISO C standard is
 40593 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

40594 CX The *iswlower()* and *iswlower_l()* functions shall test whether *wc* is a wide-character code
 40595 CX representing a character of class **lower** in the current locale, or in the locale represented by
 40596 *locale*, respectively; see XBD Chapter 7 (on page 135).

40597 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 40598 code corresponding to a valid character in the locale used by the function, or equal to the value
 40599 of the macro WEOF. If the argument has any other value, the behavior is undefined.

40600 CX The behavior is undefined if the *locale* argument to *iswlower_l()* is the special locale object
 40601 LC_GLOBAL_LOCALE or is not a valid locale object handle.

40602 **RETURN VALUE**

40603 CX The *iswlower()* and *iswlower_l()* functions shall return non-zero if *wc* is a lowercase letter wide-
 40604 character code; otherwise, they shall return 0.

40605 **ERRORS**

40606 No errors are defined.

40607 **EXAMPLES**

40608 None.

40609 **APPLICATION USAGE**

40610 To ensure applications portability, especially across natural languages, only these functions and
 40611 the functions in the reference pages listed in the SEE ALSO section should be used for character
 40612 classification.

40613 **RATIONALE**

40614 None.

40615 **FUTURE DIRECTIONS**

40616 None.

40617 **SEE ALSO**

40618 *iswalnum()*, *iswalphabeta()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswprint()*, *iswpunct()*,
 40619 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()* (on page 2205) 1

40620 XBD Chapter 7 (on page 135), [<locale.h>](#), [<wctype.h>](#)

40621 **CHANGE HISTORY**

40622 First released in Issue 4.

40623 **Issue 5**

40624 The following change has been made in this version for alignment with
 40625 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40626 The SYNOPSIS has been changed to indicate that this function and associated data types
40627 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40628 **Issue 6**

40629 The normative text is updated to avoid use of the term “must” for application requirements.

40630 **Issue 7**

40631 The `iswlower_l()` function is added from The Open Group Technical Standard, 2006, Extended
40632 API Set Part 4.

40633 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0329 [302], XSH/TC1-2008/0330 [283],
40634 and XSH/TC1-2008/0331 [283] are applied.

40635 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0188 [685] is applied.

40636 **NAME**40637 `iswprint, iswprint_l` ‡test for a printable wide-character code40638 **SYNOPSIS**40639 `#include <wctype.h>`40640 `int iswprint(wint_t wc);`40641 CX `int iswprint_l(wint_t wc, locale_t locale);`40642 **DESCRIPTION**40643 CX For `iswprint()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40644 CX The `iswprint()` and `iswprint_l()` functions shall test whether `wc` is a wide-character code representing a character of class **print** in the current locale, or in the locale represented by `locale`, respectively; see XBD Chapter 7 (on page 135).40645 CX The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.40646 CX The behavior is undefined if the `locale` argument to `iswprint_l()` is the special locale object `LC_GLOBAL_LOCALE` or is not a valid locale object handle.40647 **RETURN VALUE**40648 CX The `iswprint()` and `iswprint_l()` functions shall return non-zero if `wc` is a printable wide-character code; otherwise, they shall return 0.40649 **ERRORS**

40650 No errors are defined.

40651 **EXAMPLES**

40652 None.

40653 **APPLICATION USAGE**

40654 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40655 **RATIONALE**

40656 None.

40657 **FUTURE DIRECTIONS**

40658 None.

40659 **SEE ALSO**40660 `iswalnum()`, `iswalphabeta()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswpunct()`, `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`40661 XBD Chapter 7 (on page 135), `<locale.h>`, `<wctype.h>`40662 **CHANGE HISTORY**

40663 First released in Issue 4.

40664 **Issue 5**

40665 The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

40678 The SYNOPSIS has been changed to indicate that this function and associated data types
40679 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40680 **Issue 6**

40681 The normative text is updated to avoid use of the term “must” for application requirements.

40682 **Issue 7**

40683 The *iswprint_l()* function is added from The Open Group Technical Standard, 2006, Extended
40684 API Set Part 4.

40685 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0332 [302], XSH/TC1-2008/0333 [283],
40686 and XSH/TC1-2008/0334 [283] are applied.

40687 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0189 [685] is applied.

40688 **NAME**

40689 iswpunct, iswpunct_l ‡test for a punctuation wide-character code

40690 **SYNOPSIS**

40691 #include <wctype.h>

40692 int iswpunct(wint_t wc);

40693 CX int iswpunct_l(wint_t wc, locale_t locale);

40694 **DESCRIPTION**40695 CX For *iswpunct()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40698 CX The *iswpunct()* and *iswpunct_l()* functions shall test whether *wc* is a wide-character code representing a character of class **punct** in the current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page 135).40701 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.40704 CX The behavior is undefined if the *locale* argument to *iswpunct_l()* is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.40706 **RETURN VALUE**40707 CX The *iswpunct()* and *iswpunct_l()* functions shall return non-zero if *wc* is a punctuation wide-character code; otherwise, they shall return 0.40709 **ERRORS**

40710 No errors are defined.

40711 **EXAMPLES**

40712 None.

40713 **APPLICATION USAGE**

40714 To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

40717 **RATIONALE**

40718 None.

40719 **FUTURE DIRECTIONS**

40720 None.

40721 **SEE ALSO**40722 *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40724 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40725 **CHANGE HISTORY**

40726 First released in Issue 4.

40727 **Issue 5**

40728 The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

40730 The SYNOPSIS has been changed to indicate that this function and associated data types
40731 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40732 **Issue 6**

40733 The normative text is updated to avoid use of the term “must” for application requirements.

40734 **Issue 7**

40735 The `iswpunct_l()` function is added from The Open Group Technical Standard, 2006, Extended
40736 API Set Part 4.

40737 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0335 [302], XSH/TC1-2008/0336 [283],
40738 and XSH/TC1-2008/0337 [283] are applied.

40739 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0190 [685] is applied.

40740 **NAME**

40741 iswspace, iswspace_l ¶test for a white-space wide-character code

40742 **SYNOPSIS**

40743 #include <wctype.h>

40744 int iswspace(wint_t wc);

40745 CX int iswspace_l(wint_t wc, locale_t locale);

40746 **DESCRIPTION**40747 CX For *iswspace()*: The functionality described on this reference page is aligned with the ISO C
40748 standard. Any conflict between the requirements described here and the ISO C standard is
40749 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40750 CX The *iswspace()* and *iswspace_l()* functions shall test whether *wc* is a wide-character code
40751 CX representing a character of class **space** in the current locale, or in the locale represented by *locale*,
40752 respectively; see XBD Chapter 7 (on page 135).40753 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
40754 code corresponding to a valid character in the locale used by the function, or equal to the value
40755 of the macro WEOF. If the argument has any other value, the behavior is undefined.40756 CX The behavior is undefined if the *locale* argument to *iswspace_l()* is the special locale object
40757 LC_GLOBAL_LOCALE or is not a valid locale object handle.40758 **RETURN VALUE**40759 CX The *iswspace()* and *iswspace_l()* functions shall return non-zero if *wc* is a white-space wide-
40760 character code; otherwise, they shall return 0.40761 **ERRORS**

40762 No errors are defined.

40763 **EXAMPLES**

40764 None.

40765 **APPLICATION USAGE**40766 To ensure applications portability, especially across natural languages, only these functions and
40767 the functions in the reference pages listed in the SEE ALSO section should be used for character
40768 classification.40769 **RATIONALE**

40770 None.

40771 **FUTURE DIRECTIONS**

40772 None.

40773 **SEE ALSO**40774 *iswalnum()*, *iswalphabeta()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
40775 *iswpunct()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40776 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40777 **CHANGE HISTORY**

40778 First released in Issue 4.

40779 **Issue 5**40780 The following change has been made in this version for alignment with
40781 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40782 The SYNOPSIS has been changed to indicate that this function and associated data types
40783 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40784 **Issue 6**

40785 The normative text is updated to avoid use of the term “must” for application requirements.

40786 **Issue 7**

40787 The `iswspace_l()` function is added from The Open Group Technical Standard, 2006, Extended
40788 API Set Part 4.

40789 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0338 [302], XSH/TC1-2008/0339 [283],
40790 and XSH/TC1-2008/0340 [283] are applied.

40791 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0191 [685] is applied.

40792 **NAME**

40793 iswupper, iswupper_l — test for an uppercase letter wide-character code

40794 **SYNOPSIS**

```
40795 #include <wctype.h>
40796 int iswupper(wint_t wc);
40797 CX int iswupper_l(wint_t wc, locale_t locale);
```

40798 **DESCRIPTION**

40799 CX For *iswupper()*: The functionality described on this reference page is aligned with the ISO C
 40800 standard. Any conflict between the requirements described here and the ISO C standard is
 40801 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

40802 CX The *iswupper()* and *iswupper_l()* functions shall test whether *wc* is a wide-character code
 40803 CX representing a character of class **upper** in the current locale, or in the locale represented by
 40804 *locale*, respectively; see XBD Chapter 7 (on page 135).

40805 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 40806 code corresponding to a valid character in the locale used by the function, or equal to the value
 40807 of the macro WEOF. If the argument has any other value, the behavior is undefined.

40808 CX The behavior is undefined if the *locale* argument to *iswupper_l()* is the special locale object
 40809 LC_GLOBAL_LOCALE or is not a valid locale object handle.

40810 **RETURN VALUE**

40811 CX The *iswupper()* and *iswupper_l()* functions shall return non-zero if *wc* is an uppercase letter
 40812 wide-character code; otherwise, they shall return 0.

40813 **ERRORS**

40814 No errors are defined.

40815 **EXAMPLES**

40816 None.

40817 **APPLICATION USAGE**

40818 To ensure applications portability, especially across natural languages, only these functions and
 40819 the functions in the reference pages listed in the SEE ALSO section should be used for character
 40820 classification.

40821 **RATIONALE**

40822 None.

40823 **FUTURE DIRECTIONS**

40824 None.

40825 **SEE ALSO**

40826 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
 40827 *iswpunct()*, *iswspace()*, *iswxdigit()*, *setlocale()*, *uselocale()*

40828 XBD Chapter 7 (on page 135), [<locale.h>](#), [<wctype.h>](#)40829 **CHANGE HISTORY**

40830 First released in Issue 4.

40831 **Issue 5**

40832 The following change has been made in this version for alignment with
 40833 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40834 The SYNOPSIS has been changed to indicate that this function and associated data types
40835 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40836 **Issue 6**

40837 The normative text is updated to avoid use of the term “must” for application requirements.

40838 **Issue 7**

40839 The `iswupper_l()` function is added from The Open Group Technical Standard, 2006, Extended
40840 API Set Part 4.

40841 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0341 [302], XSH/TC1-2008/0342 [283],
40842 and XSH/TC1-2008/0343 [283] are applied.

40843 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0192 [685] is applied.

40844 **NAME**

40845 iswxdigit, iswxdigit_l ¶test for a hexadecimal digit wide-character code

40846 **SYNOPSIS**

40847 #include <wctype.h>

40848 int iswxdigit(wint_t wc);

40849 CX int iswxdigit_l(wint_t wc, locale_t locale);

40850 **DESCRIPTION**40851 CX For *iswxdigit()*: The functionality described on this reference page is aligned with the ISO C
40852 standard. Any conflict between the requirements described here and the ISO C standard is
40853 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.40854 CX The *iswxdigit()* and *iswxdigit_l()* functions shall test whether *wc* is a wide-character code
40855 CX representing a character of class **xdigit** in the current locale, or in the locale represented by
40856 *locale*, respectively; see XBD Chapter 7 (on page 135).40857 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
40858 code corresponding to a valid character in the locale used by the function, or equal to the value
40859 of the macro WEOF. If the argument has any other value, the behavior is undefined.40860 CX The behavior is undefined if the *locale* argument to *iswxdigit_l()* is the special locale object
40861 LC_GLOBAL_LOCALE or is not a valid locale object handle.40862 **RETURN VALUE**40863 CX The *iswxdigit()* and *iswxdigit_l()* functions shall return non-zero if *wc* is a hexadecimal digit
40864 wide-character code; otherwise, they shall return 0.40865 **ERRORS**

40866 No errors are defined.

40867 **EXAMPLES**

40868 None.

40869 **APPLICATION USAGE**40870 To ensure applications portability, especially across natural languages, only these functions and
40871 the functions in the reference pages listed in the SEE ALSO section should be used for character
40872 classification.40873 **RATIONALE**

40874 None.

40875 **FUTURE DIRECTIONS**

40876 None.

40877 **SEE ALSO**40878 *iswalnum()*, *iswalphalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
40879 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, *uselocale()*

40880 XBD Chapter 7 (on page 135), <locale.h>, <wctype.h>

40881 **CHANGE HISTORY**

40882 First released in Issue 4.

40883 **Issue 5**40884 The following change has been made in this version for alignment with
40885 ISO/IEC 9899:1990/Amendment 1:1995 (E):

40886 The SYNOPSIS has been changed to indicate that this function and associated data types
40887 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

40888 **Issue 6**

40889 The normative text is updated to avoid use of the term “must” for application requirements.

40890 **Issue 7**

40891 The `iswxdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended
40892 API Set Part 4.

40893 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0344 [302], XSH/TC1-2008/0345 [283],
40894 and XSH/TC1-2008/0346 [283] are applied.

40895 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0193 [685] is applied.

40896 **NAME**

40897 isxdigit, isxdigit_l ¶test for a hexadecimal digit

40898 **SYNOPSIS**

```
40899 #include <ctype.h>
40900 int isxdigit(int c);
40901 CX int isxdigit_l(int c, locale_t locale);
```

40902 **DESCRIPTION**

40903 CX For *isxdigit()*: The functionality described on this reference page is aligned with the ISO C
 40904 standard. Any conflict between the requirements described here and the ISO C standard is
 40905 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

40906 CX The *isxdigit()* and *isxdigit_l()* functions shall test whether *c* is a character of class **xdigit** in the
 40907 CX current locale, or in the locale represented by *locale*, respectively; see XBD Chapter 7 (on page
 40908 135).

40909 The *c* argument is an **int**, the value of which the application shall ensure is a character
 40910 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 40911 any other value, the behavior is undefined.

40912 CX The behavior is undefined if the *locale* argument to *isxdigit_l()* is the special locale object
 40913 LC_GLOBAL_LOCALE or is not a valid locale object handle.

40914 **RETURN VALUE**

40915 CX The *isxdigit()* and *isxdigit_l()* functions shall return non-zero if *c* is a hexadecimal digit;
 40916 otherwise, they shall return 0.

40917 **ERRORS**

40918 No errors are defined.

40919 **EXAMPLES**

40920 None.

40921 **APPLICATION USAGE**

40922 To ensure applications portability, especially across natural languages, only these functions and
 40923 the functions in the reference pages listed in the SEE ALSO section should be used for character
 40924 classification.

40925 **RATIONALE**

40926 None.

40927 **FUTURE DIRECTIONS**

40928 None.

40929 **SEE ALSO**

40930 *isalnum()*, *isalpha()*, *isblank()*, *isctrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,
 40931 *isupper()*

40932 XBD Chapter 7 (on page 135), **<ctype.h>**40933 **CHANGE HISTORY**

40934 First released in Issue 1. Derived from Issue 1 of the SVID.

40935 **Issue 6**

40936 The normative text is updated to avoid use of the term “must” for application requirements.

40937 **Issue 7**

40938 The *isxdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API
40939 Set Part 4.

40940 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0347 [302], XSH/TC1-2008/0348 [283],
40941 and XSH/TC1-2008/0349 [283] are applied.

40942 **NAME**

40943 j0, j1, jn ‡Bessel functions of the first kind

40944 **SYNOPSIS**

```
40945 XSI      #include <math.h>
40946         double j0(double x);
40947         double j1(double x);
40948         double jn(int n, double x);
```

40949 **DESCRIPTION**

40950 The *j0()*, *j1()*, and *jn()* functions shall compute Bessel functions of *x* of the first kind of orders 0,
40951 1, and *n*, respectively.

40952 An application wishing to check for error situations should set *errno* to zero and call
40953 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
40954 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
40955 zero, an error has occurred.

40956 **RETURN VALUE**

40957 Upon successful completion, these functions shall return the relevant Bessel value of *x* of the
40958 first kind.

40959 If the *x* argument is too large in magnitude, or the correct result would cause underflow, 0 shall
40960 be returned and a range error may occur.

40961 MXX If *x* is NaN, a NaN shall be returned.

40962 **ERRORS**

40963 These functions may fail if:

40964 Range Error The value of *x* was too large in magnitude, or an underflow occurred.

40965 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
40966 then *errno* shall be set to [ERANGE]. If the integer expression
40967 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
40968 floating-point exception shall be raised.

40969 No other errors shall occur.

40970 **EXAMPLES**

40971 None.

40972 **APPLICATION USAGE**

40973 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
40974 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

40975 **RATIONALE**

40976 None.

40977 **FUTURE DIRECTIONS**

40978 None.

40979 **SEE ALSO**

40980 *feclearexcept()*, *fetestexcept()*, *isnan()*, *y0()*

40981 XBD Section 4.20 (on page 117), <math.h>

40982 **CHANGE HISTORY**

40983 First released in Issue 1. Derived from Issue 1 of the SVID.

40984 **Issue 5**

40985 The DESCRIPTION is updated to indicate how an application should check for an error. This
40986 text was previously published in the APPLICATION USAGE section.

40987 **Issue 6**

40988 The may fail [EDOM] error is removed for the case for NaN.

40989 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
40990 with the ISO/IEC 9899:1999 standard.

40991 **Issue 7**

40992 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0350 [68] is applied.

40993 **NAME**

40994 jrand48 ‡generate a uniformly distributed pseudo-random long signed integer

40995 **SYNOPSIS**

```
40996 XSI       #include <stdlib.h>  
40997       long jrand48(unsigned short xsubi[3]);
```

40998 **DESCRIPTION**40999 Refer to *drand48()*.

41000 **NAME**

41001 kill — send a signal to a process or a group of processes

41002 **SYNOPSIS**

```
41003 CX #include <signal.h>
41004 int kill(pid_t pid, int sig);
```

41005 **DESCRIPTION**

41006 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The
 41007 signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is
 41008 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can
 41009 be used to check the validity of *pid*.

41010 For a process to have permission to send a signal to a process designated by *pid*, unless the
 41011 sending process has appropriate privileges, the real or effective user ID of the sending process
 41012 shall match the real or saved set-user-ID of the receiving process.

41013 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

41014 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)
 41015 whose process group ID is equal to the process group ID of the sender, and for which the process
 41016 has permission to send a signal.

41017 If *pid* is -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for
 41018 which the process has permission to send that signal.

41019 If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of
 41020 system processes) whose process group ID is equal to the absolute value of *pid*, and for which
 41021 the process has permission to send a signal.

41022 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for
 41023 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function
 41024 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending
 41025 thread before *kill()* returns.

41026 The user ID tests described above shall not be applied when sending SIGCONT to a process that
 41027 is a member of the same session as the sending process.

41028 An implementation that provides extended security controls may impose further
 41029 implementation-defined restrictions on the sending of signals, including the null signal. In
 41030 particular, the system may deny the existence of some or all of the processes specified by *pid*.

41031 The *kill()* function is successful if the process has permission to send *sig* to any of the processes
 41032 specified by *pid*. If *kill()* fails, no signal shall be sent.

41033 **RETURN VALUE**

41034 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 41035 indicate the error.

41036 **ERRORS**

41037 The *kill()* function shall fail if:

- | | | |
|-------|----------|---|
| 41038 | [EINVAL] | The value of the <i>sig</i> argument is an invalid or unsupported signal number. |
| 41039 | [EPERM] | The process does not have permission to send the signal to any receiving process. |
| 41040 | | |

41041 [ESRCH] No process or process group can be found corresponding to that specified by
41042 *pid*.

41043 EXAMPLES

41044 None.

41045 APPLICATION USAGE

41046 None.

41047 RATIONALE

41048 The semantics for permission checking for *kill()* differed between System V and most other
41049 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of
41050 POSIX.1-2017 agree with System V. Specifically, a set-user-ID process cannot protect itself
41051 against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice
41052 allows the user who starts an application to send it signals even if it changes its effective user ID.
41053 The other semantics give more power to an application that wants to protect itself from the user
41054 who ran it.

41055 Some implementations provide semantic extensions to the *kill()* function when the absolute
41056 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a
41057 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not
41058 included in this volume of POSIX.1-2017, although a conforming implementation could provide
41059 such an extension.

41060 The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

41061 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the
41062 calling process and that signal is not blocked, that signal would be delivered before *kill()*
41063 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.
41064 However, historical implementations that provide only the *signal()* function make only the
41065 weaker guarantee in this volume of POSIX.1-2017, because they only deliver one signal each
41066 time a process enters the kernel. Modifications to such implementations to support the
41067 *sigaction()* function generally require entry to the kernel following return from a signal-catching
41068 function, in order to restore the signal mask. Such modifications have the effect of satisfying the
41069 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.
41070 The standard developers considered making the stronger requirement except when *signal()* is
41071 used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the
41072 stronger requirement whenever possible. In practice, the weaker requirement is the same, except
41073 in the rare case when two signals arrive during a very short window. This reasoning also applies
41074 to a similar requirement for *sigprocmask()*.

41075 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID
41076 security checks. This allows a job control shell to continue a job even if processes in the job have
41077 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of
41078 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any
41079 process in the same session regardless of user ID security checks. This is less restrictive than BSD
41080 in the sense that ancestor processes (in the same session) can now be the recipient. It is more
41081 restrictive than BSD in the sense that descendant processes that form new sessions are now
41082 subject to the user ID checks. A similar relaxation of security is not necessary for the other job
41083 control signals since those signals are typically sent by the terminal driver in recognition of
41084 special characters being typed; the terminal driver bypasses all security checks.

41085 In secure implementations, a process may be restricted from sending a signal to a process having
41086 a different security label. In order to prevent the existence or nonexistence of a process from
41087 being used as a covert channel, such processes should appear nonexistent to the sender; that is,
41088 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

41089 Historical implementations varied on the result of a *kill()* with *pid* indicating a zombie process.
 41090 Some indicated success on such a call (subject to permission checking), while others gave an
 41091 error of [ESRCH]. Since the definition of process lifetime in this volume of POSIX.1-2017 covers
 41092 zombie processes, the [ESRCH] error as described is inappropriate in this case and
 41093 implementations that give this error do not conform. This means that an application cannot have
 41094 a parent process check for termination of a particular child by sending it the null signal with
 41095 *kill()*, but must instead use *waitpid()* or *waitid()*.

41096 There is some belief that the name *kill()* is misleading, since the function is not always intended
 41097 to cause process termination. However, the name is common to all historical implementations,
 41098 and any change would be in conflict with the goal of minimal changes to existing application
 41099 code.

41100 FUTURE DIRECTIONS

41101 None.

41102 SEE ALSO

41103 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, *wait()*

41104 XBD [<signal.h>](#), [<sys/types.h>](#)

41105 CHANGE HISTORY

41106 First released in Issue 1. Derived from Issue 1 of the SVID.

41107 Issue 5

41108 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

41109 Issue 6

41110 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

41111 The following new requirements on POSIX implementations derive from alignment with the
 41112 Single UNIX Specification:

41113 In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-
 41114 user-ID of the calling process is checked in place of its effective user ID. This is a FIPS
 41115 requirement.

41116 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 41117 required for conforming implementations of previous POSIX specifications, it was not
 41118 required for UNIX applications.

41119 The behavior when *pid* is -1 is now specified. It was previously explicitly unspecified in
 41120 the POSIX.1-1988 standard.

41121 The normative text is updated to avoid use of the term “must” for application requirements.

41122 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/51 is applied, correcting the RATIONALE
 41123 section.

41124 Issue 7

41125 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0194 [765] is applied.

41126 **NAME**

41127 killpg — send a signal to a process group

41128 **SYNOPSIS**

```
41129 XSI #include <signal.h>
41130 int killpg(pid_t pgrp, int sig);
```

41131 **DESCRIPTION**41132 The *killpg()* function shall send the signal specified by *sig* to the process group specified by *pgrp*.

41133 If *pgrp* is greater than 1, *killpg(pgrp, sig)* shall be equivalent to *kill(-pgrp, sig)*. If *pgrp* is less than or
 41134 equal to 1, the behavior of *killpg()* is undefined.

41135 **RETURN VALUE**41136 Refer to *kill()*.41137 **ERRORS**41138 Refer to *kill()*.41139 **EXAMPLES**41140 **Sending a Signal to All Other Members of a Process Group**

41141 The following example shows how the calling process could send a signal to all other members
 41142 of its process group. To prevent itself from receiving the signal it first makes itself immune to the
 41143 signal by ignoring it.

```
41144 #include <signal.h>
41145 #include <unistd.h>
41146 ...
41147     if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
41148         /* Handle error */;
41149     if (killpg(getpgrp(), SIGUSR1) == -1)
41150         /* Handle error */;
```

41151 **APPLICATION USAGE**

41152 None.

41153 **RATIONALE**

41154 None.

41155 **FUTURE DIRECTIONS**

41156 None.

41157 **SEE ALSO**41158 *getpgid()*, *getpid()*, *kill()*, *raise()*41159 XBD [<signal.h>](#)41160 **CHANGE HISTORY**

41161 First released in Issue 4, Version 2.

41162 **Issue 5**

41163 Moved from X/OPEN UNIX extension to BASE.

41164 **Issue 6**

41165 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/52 is applied, adding the example to the
41166 EXAMPLES section.

41167 **NAME**

41168 l64a ¶convert a 32-bit integer to a radix-64 ASCII string

41169 **SYNOPSIS**

```
41170 XSI        #include <stdlib.h>
41171        char *l64a(long value);
```

41172 **DESCRIPTION**41173 Refer to [a64l\(\)](#).

41174 **NAME**

41175 labs, llabs — return a long integer absolute value

41176 **SYNOPSIS**

41177 #include <stdlib.h>

41178 long labs(long i);

41179 long long llabs(long long i);

41180 **DESCRIPTION**

41181 CX The functionality described on this reference page is aligned with the ISO C standard. Any
41182 conflict between the requirements described here and the ISO C standard is unintentional. This
41183 volume of POSIX.1-2017 defers to the ISO C standard.

41184 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*
41185 function shall compute the absolute value of the **long long** integer operand *i*. If the result
41186 cannot be represented, the behavior is undefined.

41187 **RETURN VALUE**41188 The *labs()* function shall return the absolute value of the **long** integer operand.41189 The *llabs()* function shall return the absolute value of the **long long** integer operand.41190 **ERRORS**

41191 No errors are defined.

41192 **EXAMPLES**

41193 None.

41194 **APPLICATION USAGE**

41195 None.

41196 **RATIONALE**

41197 None.

41198 **FUTURE DIRECTIONS**

41199 None.

41200 **SEE ALSO**41201 [abs\(\)](#)41202 XBD [<stdlib.h>](#)41203 **CHANGE HISTORY**

41204 First released in Issue 4. Derived from the ISO C standard.

41205 **Issue 6**41206 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.41207 **Issue 7**

41208 SD5-XSH-ERN-152 is applied, correcting the RETURN VALUE section.

41209 **NAME**

41210 lchown — change the owner and group of a symbolic link

41211 **SYNOPSIS**

41212 #include <unistd.h>

41213 int lchown(const char *path, uid_t owner, gid_t group);

41214 **DESCRIPTION**

41215 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a
 41216 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,
 41217 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

41218 **RETURN VALUE**

41219 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 41220 indicate an error.

41221 **ERRORS**41222 The *lchown()* function shall fail if:41223 [EACCES] Search permission is denied on a component of the path prefix of *path*.

41224 [EINVAL] The owner or group ID is not a value supported by the implementation.

41225 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 41226 argument.

41227 [ENAMETOOLONG]

41228 The length of a component of a pathname is longer than {NAME_MAX}.

41229 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

41230 [ENOTDIR] A component of the path prefix names an existing file that is neither a
 41231 directory nor a symbolic link to a directory, or the *path* argument contains at
 41232 least one non-*<slash>* character and ends with one or more trailing *<slash>*
 41233 characters and the last pathname component names an existing file that is
 41234 neither a directory nor a symbolic link to a directory.

41235 [EPERM] The effective user ID does not match the owner of the file and the process does
 41236 not have appropriate privileges.

41237 [EROFS] The file resides on a read-only file system.

41238 The *lchown()* function may fail if:

41239 [EIO] An I/O error occurred while reading or writing to the file system.

41240 [EINTR] A signal was caught during execution of the function.

41241 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 41242 resolution of the *path* argument.

41243 [ENAMETOOLONG]

41244 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 41245 symbolic link produced an intermediate result with a length that exceeds
 41246 {PATH_MAX}.

41247 **EXAMPLES**41248 **Changing the Current Owner of a File**

41249 The following example shows how to change the ownership of the symbolic link named
41250 **/modules/pass1** to the user ID associated with `“jones”` and the group ID associated with `“cnd”`.

41251 The numeric value for the user ID is obtained by using the `getpwnam()` function. The numeric
41252 value for the group ID is obtained by using the `getgrnam()` function.

```
41253 #include <sys/types.h>
41254 #include <unistd.h>
41255 #include <pwd.h>
41256 #include <grp.h>

41257 struct passwd *pwd;
41258 struct group *grp;
41259 char *path = "/modules/pass1";
41260 ...
41261 pwd = getpwnam("jones");
41262 grp = getgrnam("cnd");
41263 lchown(path, pwd->pw_uid, grp->gr_gid);
```

41264 **APPLICATION USAGE**

41265 On implementations which support symbolic links as directory entries rather than files, `lchown()`
41266 may fail.

41267 **RATIONALE**

41268 None.

41269 **FUTURE DIRECTIONS**

41270 None.

41271 **SEE ALSO**

41272 [*chown\(\)*](#), [*symlink\(\)*](#)

41273 XBD [**<unistd.h>**](#)

41274 **CHANGE HISTORY**

41275 First released in Issue 4, Version 2.

41276 **Issue 5**

41277 Moved from X/OPEN UNIX extension to BASE.

41278 **Issue 6**

41279 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
41280 [ELOOP] error condition is added.

41281 The Open Group Base Resolution bwg2001-013 is applied, adding wording to the
41282 APPLICATION USAGE.

41283 **Issue 7**

41284 Austin Group Interpretation 1003.1-2001 #143 is applied.

41285 The `lchown()` function is moved from the XSI option to the Base.

41286 The [EOPNOTSUPP] error is removed.

41287 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
41288 pathname exists but is not a directory or a symbolic link to a directory.

41289

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0351 [324] is applied.

41290 **NAME**

41291 lcong48 ¶seed a uniformly distributed pseudo-random signed long integer generator

41292 **SYNOPSIS**

```
41293 XSI #include <stdlib.h>  
41294 void lcong48(unsigned short param[7]);
```

41295 **DESCRIPTION**

41296 Refer to *drand48()*.

41297 **NAME**41298 ldexp, ldexpf, ldexpl \uparrow load exponent of a floating-point number41299 **SYNOPSIS**

```
41300 #include <math.h>
41301 double ldexp(double x, int exp);
41302 float ldexpf(float x, int exp);
41303 long double ldexpl(long double x, int exp);
```

41304 **DESCRIPTION**

41305 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41306 conflict between the requirements described here and the ISO C standard is unintentional. This
 41307 volume of POSIX.1-2017 defers to the ISO C standard.

41308 These functions shall compute the quantity $x * 2^{exp}$.

41309 An application wishing to check for error situations should set *errno* to zero and call
 41310 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41311 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41312 zero, an error has occurred.

41313 **RETURN VALUE**41314 Upon successful completion, these functions shall return x multiplied by 2, raised to the power
41315 exp .

41316 If these functions would cause overflow, a range error shall occur and *ldexp*(\cdot), *ldexpf*(\cdot), and
 41317 *ldexpl*(\cdot) shall return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL (according to the sign of
 41318 x), respectively.

41319 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 41320 MXX and *ldexp*(\cdot), *ldexpf*(\cdot), and *ldexpl*(\cdot) shall return 0.0, or (if IEC 60559 Floating-Point is not
 41321 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 41322 FLT_MIN, and LDBL_MIN, respectively.

41323 MX If x is NaN, a NaN shall be returned.41324 If x is ± 0 or \pm Inf, x shall be returned.41325 If exp is 0, x shall be returned.

41326 MXX If the correct value would cause underflow, and is representable, a range error may occur and
 41327 the correct value shall be returned.

41328 **ERRORS**

41329 These functions shall fail if:

41330 Range Error The result overflows.

41331 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41332 then *errno* shall be set to [ERANGE]. If the integer expression
 41333 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 41334 floating-point exception shall be raised.

41335 These functions may fail if:

41336 Range Error The result underflows.

41337 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41338 then *errno* shall be set to [ERANGE]. If the integer expression
 41339 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

41340 floating-point exception shall be raised.

41341 **EXAMPLES**

41342 None.

41343 **APPLICATION USAGE**

41344 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
41345 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41346 **RATIONALE**

41347 None.

41348 **FUTURE DIRECTIONS**

41349 None.

41350 **SEE ALSO**

41351 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [frexp\(\)](#), [isnan\(\)](#)

41352 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

41353 **CHANGE HISTORY**

41354 First released in Issue 1. Derived from Issue 1 of the SVID.

41355 **Issue 5**

41356 The DESCRIPTION is updated to indicate how an application should check for an error. This
41357 text was previously published in the APPLICATION USAGE section.

41358 **Issue 6**

41359 The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999
41360 standard.

41361 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
41362 revised to align with the ISO/IEC 9899:1999 standard.

41363 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
41364 marked.

41365 **Issue 7**

41366 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0352 [68] and XSH/TC1-2008/0353
41367 [68] are applied.

41368 **NAME**

41369 ldiv, lldiv — compute quotient and remainder of a long division

41370 **SYNOPSIS**

41371 #include <stdlib.h>

41372 ldiv_t ldiv(long numer, long denom);

41373 lldiv_t lldiv(long long numer, long long denom);

41374 **DESCRIPTION**

41375 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41376 conflict between the requirements described here and the ISO C standard is unintentional. This
 41377 volume of POSIX.1-2017 defers to the ISO C standard.

41378 These functions shall compute the quotient and remainder of the division of the numerator
 41379 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**
 41380 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude
 41381 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is
 41382 undefined; otherwise, *quot * denom + rem* shall equal *numer*.

41383 **RETURN VALUE**

41384 The *ldiv()* function shall return a structure of type **ldiv_t**, comprising both the quotient and the
 41385 remainder. The structure shall include the following members, in any order:

```
41386 long    quot;    /* Quotient */
41387 long    rem;     /* Remainder */
```

41388 The *lldiv()* function shall return a structure of type **lldiv_t**, comprising both the quotient and the
 41389 remainder. The structure shall include the following members, in any order:

```
41390 long long quot;    /* Quotient */
41391 long long rem;     /* Remainder */
```

41392 **ERRORS**

41393 No errors are defined.

41394 **EXAMPLES**

41395 None.

41396 **APPLICATION USAGE**

41397 None.

41398 **RATIONALE**

41399 None.

41400 **FUTURE DIRECTIONS**

41401 None.

41402 **SEE ALSO**41403 [div\(\)](#)41404 XBD [<stdlib.h>](#)41405 **CHANGE HISTORY**

41406 First released in Issue 4. Derived from the ISO C standard.

41407 **Issue 6**41408 The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.

41409 **NAME**

41410 lfind — find entry in a linear search table

41411 **SYNOPSIS**

```
41412 XSI #include <search.h>
41413 void *lfind(const void *key, const void *base, size_t *nelp,
41414            size_t width, int (*compar)(const void *, const void *));
```

41415 **DESCRIPTION**

41416 Refer to [lsearch\(\)](#).

41417 **NAME**

41418 lgamma, lgammaf, lgammal, signgam ‡log gamma function

41419 **SYNOPSIS**

```
41420       #include <math.h>
41421       double lgamma(double x);
41422       float lgammaf(float x);
41423       long double lgammal(long double x);
41424 XSI     extern int signgam;
```

41425 **DESCRIPTION**

41426 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41427 conflict between the requirements described here and the ISO C standard is unintentional. This
 41428 volume of POSIX.1-2017 defers to the ISO C standard.

41429 These functions shall compute $\log_e |\Gamma(x)|$ where $\Gamma(x)$ is defined as $\int_0^{\infty} e^{-t} t^{x-1} dt$. The argument x
 41430 need not be a non-positive integer ($\Gamma(x)$ is defined over the reals, except the non-positive
 41431 integers).

41432 XSI If x is NaN, $-\text{Inf}$, or a negative integer, the value of *signgam* is unspecified.

41433 CX These functions need not be thread-safe.

41434 An application wishing to check for error situations should set *errno* to zero and call
 41435 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41436 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41437 zero, an error has occurred.

41438 **RETURN VALUE**

41439 Upon successful completion, these functions shall return the logarithmic gamma of x .

41440 If x is a non-positive integer, a pole error shall occur and *lgamma()*, *lgammaf()*, and *lgammal()*
 41441 shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.

41442 If the correct value would cause overflow, a range error shall occur and *lgamma()*, *lgammaf()*,
 41443 and *lgammal()* shall return $\pm\text{HUGE_VAL}$, $\pm\text{HUGE_VALF}$, and $\pm\text{HUGE_VALL}$ (having the same
 41444 sign as the correct value), respectively.

41445 MX If x is NaN, a NaN shall be returned.

41446 If x is 1 or 2, +0 shall be returned.

41447 If x is $\pm\text{Inf}$, +Inf shall be returned.

41448 **ERRORS**

41449 These functions shall fail if:

41450 Pole Error The x argument is a negative integer or zero.

41451 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41452 then *errno* shall be set to [ERANGE]. If the integer expression
 41453 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 41454 floating-point exception shall be raised.

41455 Range Error The result overflows.

41456 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41457 then *errno* shall be set to [ERANGE]. If the integer expression

41458 (math_errhandling & MATH_ERREXCEPT) is non-zero, then the overflow
41459 floating-point exception shall be raised.

41460 EXAMPLES

41461 None.

41462 APPLICATION USAGE

41463 On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling &
41464 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41465 RATIONALE

41466 None.

41467 FUTURE DIRECTIONS

41468 None.

41469 SEE ALSO

41470 [exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

41471 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

41472 CHANGE HISTORY

41473 First released in Issue 3.

41474 Issue 5

41475 The DESCRIPTION is updated to indicate how an application should check for an error. This
41476 text was previously published in the APPLICATION USAGE section.

41477 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

41478 Issue 6

41479 The [lgamma\(\)](#) function is no longer marked as an extension.

41480 The [lgammaf\(\)](#) and [lgammal\(\)](#) functions are added for alignment with the ISO/IEC 9899:1999
41481 standard.

41482 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
41483 revised to align with the ISO/IEC 9899:1999 standard.

41484 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
41485 marked.

41486 Functionality relating to the XSI option is marked.

41487 Issue 7

41488 Austin Group Interpretation 1003.1-2001 #156 is applied.

41489 The DESCRIPTION is clarified regarding the value of *signgam* when *x* is Nan, -Inf, or a negative
41490 integer.

41491 **NAME**

41492 link, linkat ‡link one file to another file

41493 **SYNOPSIS**

41494 #include <unistd.h>

41495 int link(const char *path1, const char *path2);

41496 OH #include <fcntl.h>

41497 int linkat(int fd1, const char *path1, int fd2,

41498 const char *path2, int flag);

41499 **DESCRIPTION**41500 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.41501 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to
41502 a pathname naming the new directory entry to be created. The *link()* function shall atomically
41503 create a new link for the existing file and the link count of the file shall be incremented by one.41504 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the
41505 implementation supports using *link()* on directories.41506 If *path1* names a symbolic link, it is implementation-defined whether *link()* follows the symbolic
41507 link, or creates a new link to the symbolic link itself.41508 Upon successful completion, *link()* shall mark for update the last file status change timestamp of
41509 the file. Also, the last data modification and last file status change timestamps of the directory
41510 that contains the new entry shall be marked for update.41511 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.41512 The implementation may require that the calling process has permission to access the existing
41513 file.41514 The *linkat()* function shall be equivalent to the *link()* function except that symbolic links shall be
41515 handled as specified by the value of *flag* (see below) and except in the case where either *path1* or
41516 *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the
41517 directory associated with the file descriptor *fd1* instead of the current working directory and
41518 similarly for *path2* and the file descriptor *fd2*. If the access mode of the open file description
41519 associated with the file descriptor is not O_SEARCH, the function shall check whether directory
41520 searches are permitted using the current permissions of the directory underlying the file
41521 descriptor. If the access mode is O_SEARCH, the function shall not perform the check.41522 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
41523 in <fcntl.h>:

41524 AT_SYMLINK_FOLLOW

41525 If *path1* names a symbolic link, a new link for the target of the symbolic link is created.41526 If *linkat()* is passed the special value AT_FDCWD in the *fd1* or *fd2* parameter, the current
41527 working directory shall be used for the respective *path* argument. If both *fd1* and *fd2* have value
41528 AT_FDCWD, the behavior shall be identical to a call to *link()*, except that symbolic links shall be
41529 handled as specified by the value of *flag*.41530 If the AT_SYMLINK_FOLLOW flag is clear in the *flag* argument and the *path1* argument names a
41531 symbolic link, a new link is created for the symbolic link *path1* and not its target.

41532 **RETURN VALUE**

41533 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 41534 return -1 and set *errno* to indicate the error.

41535 **ERRORS**

41536 These functions shall fail if:

41537 [EACCES] A component of either path prefix denies search permission, or the requested
 41538 link requires writing in a directory that denies write permission, or the calling
 41539 process does not have permission to access the existing file and this is required
 41540 by the implementation.

41541 [EEXIST] The *path2* argument resolves to an existing directory entry or refers to a
 41542 symbolic link.

41543 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path1* or
 41544 *path2* argument.

41545 [EMLINK] The number of links to the file named by *path1* would exceed {LINK_MAX}.

41546 [ENAMETOOLONG]
 41547 The length of a component of a pathname is longer than {NAME_MAX}.

41548 [ENOENT] A component of either path prefix does not exist; the file named by *path1* does
 41549 not exist; or *path1* or *path2* points to an empty string.

41550 [ENOENT] or [ENOTDIR]
 41551 The *path1* argument names an existing non-directory file, and the *path2*
 41552 argument contains at least one non-`<slash>` character and ends with one or
 41553 more trailing `<slash>` characters. If *path2* without the trailing `<slash>`
 41554 characters would name an existing file, an [ENOENT] error shall not occur.

41555 [ENOSPC] The directory to contain the link cannot be extended.

41556 [ENOTDIR] A component of either path prefix names an existing file that is neither a
 41557 directory nor a symbolic link to a directory, or the *path1* argument contains at
 41558 least one non-`<slash>` character and ends with one or more trailing `<slash>`
 41559 characters and the last pathname component names an existing file that is
 41560 neither a directory nor a symbolic link to a directory, or the *path1* argument
 41561 names an existing non-directory file and the *path2* argument names a
 41562 nonexistent file, contains at least one non-`<slash>` character, and ends with
 41563 one or more trailing `<slash>` characters.

41564 [EPERM] The file named by *path1* is a directory and either the calling process does not
 41565 have appropriate privileges or the implementation prohibits using *link()* on
 41566 directories.

41567 [EROFS] The requested link requires writing in a directory on a read-only file system.

41568 [EXDEV] The link named by *path2* and the file named by *path1* are on different file
 41569 systems and the implementation does not support links between file systems.

41570 OB XSR [EXDEV] *path1* refers to a named STREAM.

41571 The *linkat()* function shall fail if:

41572 [EACCES] The access mode of the open file description associated with *fd1* or *fd2* is not
 41573 O_SEARCH and the permissions of the directory underlying *fd1* or *fd2*,
 41574 respectively, do not permit directory searches.

- 41575 [EBADF] The *path1* or *path2* argument does not specify an absolute path and the *fd1* or
 41576 *fd2* argument, respectively, is neither AT_FDCWD nor a valid file descriptor
 41577 open for reading or searching.
- 41578 [ENOTDIR] The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*,
 41579 respectively, is a file descriptor associated with a non-directory file.
- 41580 These functions may fail if:
- 41581 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 41582 resolution of the *path1* or *path2* argument.
- 41583 [ENAMETOOLONG]
 41584 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 41585 symbolic link produced an intermediate result with a length that exceeds
 41586 {PATH_MAX}.
- 41587 The *linkat()* function may fail if:
- 41588 [EINVAL] The value of the *flag* argument is not valid.

41589 EXAMPLES

41590 Creating a Link to a File

41591 The following example shows how to create a link to a file named **/home/cnd/mod1** by creating
 41592 a new directory entry named **/modules/pass1**.

```
41593 #include <unistd.h>
41594 char *path1 = "/home/cnd/mod1";
41595 char *path2 = "/modules/pass1";
41596 int status;
41597 ...
41598 status = link (path1, path2);
```

41599 Creating a Link to a File Within a Program

41600 In the following program example, the *link()* function links the **/etc/passwd** file (defined as
 41601 **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the
 41602 current password file. Then, after removing the current password file (defined as
 41603 **PASSWDFILE**), the new password file is saved as the current password file using the *link()*
 41604 function again.

```
41605 #include <unistd.h>
41606 #define LOCKFILE "/etc/ptmp"
41607 #define PASSWDFILE "/etc/passwd"
41608 #define SAVEFILE "/etc/opasswd"
41609 ...
41610 /* Save current password file */
41611 link (PASSWDFILE, SAVEFILE);
41612 /* Remove current password file. */
41613 unlink (PASSWDFILE);
41614 /* Save new password file as current password file. */
41615 link (LOCKFILE, PASSWDFILE);
```

41616 APPLICATION USAGE

41617 Some implementations do allow links between file systems.

41618 If *path1* refers to a symbolic link, application developers should use *linkat()* with appropriate
41619 flags to select whether or not the symbolic link should be resolved.

41620 RATIONALE

41621 Linking to a directory is restricted to the superuser in most historical implementations because
41622 this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This
41623 volume of POSIX.1-2017 continues that philosophy by prohibiting *link()* and *unlink()* from
41624 doing this. Other functions could do it if the implementor designed such an extension.

41625 Some historical implementations allow linking of files on different file systems. Wording was
41626 added to explicitly allow this optional behavior.

41627 The exception for cross-file system links is intended to apply only to links that are
41628 programmatically indistinguishable from “hard” links.

41629 The purpose of the *linkat()* function is to link files in directories other than the current working
41630 directory without exposure to race conditions. Any part of the path of a file could be changed in
41631 parallel to a call to *link()*, resulting in unspecified behavior. By opening a file descriptor for the
41632 directory of both the existing file and the target location and using the *linkat()* function it can be
41633 guaranteed that the both filenames are in the desired directories.

41634 The AT_SYMLINK_FOLLOW flag allows for implementing both common behaviors of the
41635 *link()* function. The POSIX specification requires that if *path1* is a symbolic link, a new link for
41636 the target of the symbolic link is created. Many systems by default or as an alternative provide a
41637 mechanism to avoid the implicit symbolic link lookup and create a new link for the symbolic
41638 link itself.

41639 Earlier versions of this standard specified only the *link()* function, and required it to behave like
41640 *linkat()* with the AT_SYMLINK_FOLLOW flag. However, historical practice from SVR4 and
41641 Linux kernels had *link()* behaving like *linkat()* with no flags, and many systems that attempted
41642 to provide a conforming *link()* function did so in a way that was rarely used, and when it was
41643 used did not conform to the standard (e.g., by not being atomic, or by dereferencing the
41644 symbolic link incorrectly). Since applications could not rely on *link()* following links in practice,
41645 the *linkat()* function was added taking a flag to specify the desired behavior for the application.

41646 FUTURE DIRECTIONS

41647 None.

41648 SEE ALSO

41649 *rename()*, *symlink()*, *unlink()*

41650 XBD <fcntl.h>, <unistd.h>

41651 CHANGE HISTORY

41652 First released in Issue 1. Derived from Issue 1 of the SVID.

41653 Issue 6

41654 The following new requirements on POSIX implementations derive from alignment with the
41655 Single UNIX Specification:

41656 The [ELOOP] mandatory error condition is added.

41657 A second [ENAMETOOLONG] is added as an optional error condition.

41658 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41659 An explanation is added of the action when *path2* refers to a symbolic link.
- 41660 The [ELOOP] optional error condition is added.
- 41661 **Issue 7**
- 41662 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 41663 SD5-XSH-ERN-93 is applied, adding RATIONALE.
- 41664 The *linkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set
41665 Part 2.
- 41666 Functionality relating to XSI STREAMS is marked obsolescent.
- 41667 Changes are made related to support for finegrained timestamps.
- 41668 The [EOPNOTSUPP] error is removed.
- 41669 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0354 [326], XSH/TC1-2008/0355 [461],
41670 XSH/TC1-2008/0356 [326], XSH/TC1-2008/0357 [324], XSH/TC1-2008/0358 [147,429],
41671 XSH/TC1-2008/0359 [277], XSH/TC1-2008/0360 [278], and XSH/TC1-2008/0361 [278] are
41672 applied.
- 41673 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0195 [873], XSH/TC2-2008/0196 [591],
41674 XSH/TC2-2008/0197 [817], XSH/TC2-2008/0198 [822], and XSH/TC2-2008/0199 [817] are
41675 applied.

41676 **NAME**

41677 lio_listio — list directed I/O

41678 **SYNOPSIS**

41679 #include <aio.h>

41680 int lio_listio(int mode, struct aiocb *restrict const list[restrict],
41681 int nent, struct sigevent *restrict sig);41682 **DESCRIPTION**41683 The *lio_listio()* function shall initiate a list of I/O requests with a single function call.41684 The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in <aio.h> and
41685 determines whether the function returns when the I/O operations have been completed, or as
41686 soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall
41687 wait until all I/O is complete and the *sig* argument shall be ignored.41688 If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous
41689 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If
41690 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous
41691 notification occurs as specified in [Section 2.4.1](#) (on page 488) when all the requests in *list* have
41692 completed.41693 The I/O requests enumerated by *list* are submitted in an unspecified order.41694 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.
41695 The array may contain NULL elements, which shall be ignored.41696 If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list*
41697 become illegal addresses before all asynchronous I/O completed and, if necessary, the
41698 notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio_buf*
41699 member of the **aiocb** structure pointed to by the elements of the array *list* become illegal
41700 addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed,
41701 the behavior is undefined.41702 The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The
41703 supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in
41704 <aio.h>. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode*
41705 element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read()*
41706 with the *aio_cbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal to
41707 LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write()* with the *aio_cbp*
41708 equal to the address of the **aiocb** structure.41709 The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.41710 The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.41711 The *aio_nbytes* member specifies the number of bytes of data to be transferred.41712 The members of the **aiocb** structure further describe the I/O operation to be performed, in a
41713 manner identical to that of the corresponding **aiocb** structure when used by the *aio_read()* and
41714 *aio_write()* functions.41715 The *nent* argument specifies how many elements are members of the list; that is, the length of the
41716 array.41717 The behavior of this function is altered according to the definitions of synchronized I/O data
41718 integrity completion and synchronized I/O file integrity completion if synchronized I/O is
41719 enabled on the file associated with *aio_fildes*.

41720 For regular files, no data transfer shall occur past the offset maximum established in the open
41721 file description associated with *aiocbp* ~~*aio_fildes*~~.

41722 If *sig* ~~*sigev_notify*~~ is SIGEV_THREAD and *sig* ~~*sigev_notify_attributes*~~ is a non-null pointer and
41723 the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O
41724 being completed, then the behavior is undefined.

41725 RETURN VALUE

41726 If the *mode* argument has the value LIO_NOWAIT, the *lio_listio()* function shall return the value
41727 zero if the I/O operations are successfully queued; otherwise, the function shall return the value
41728 -1 and set *errno* to indicate the error.

41729 If the *mode* argument has the value LIO_WAIT, the *lio_listio()* function shall return the value zero
41730 when all the indicated I/O has completed successfully. Otherwise, *lio_listio()* shall return a value
41731 of -1 and set *errno* to indicate the error.

41732 In either case, the return value only indicates the success or failure of the *lio_listio()* call itself,
41733 not the status of the individual I/O requests. In some cases one or more of the I/O requests
41734 contained in the list may fail. Failure of an individual request does not prevent completion of
41735 any other individual request. To determine the outcome of each I/O request, the application
41736 shall examine the error status associated with each **aiocb** control block. The error statuses so
41737 returned are identical to those returned as the result of an *aio_read()* or *aio_write()* function.

41738 ERRORS

41739 The *lio_listio()* function shall fail if:

41740 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The
41741 application may check the error status for each **aiocb** to determine the
41742 individual request(s) that failed.

41743 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit
41744 {AIO_MAX} to be exceeded.

41745 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than
41746 {AIO_LISTIO_MAX}.

41747 [EINTR] A signal was delivered while waiting for all I/O requests to complete during
41748 an LIO_WAIT operation. Note that, since each I/O operation invoked by
41749 *lio_listio()* may possibly provoke a signal when it completes, this error return
41750 may be caused by the completion of one (or more) of the very I/O operations
41751 being awaited. Outstanding I/O requests are not canceled, and the application
41752 shall examine each list element to determine whether the request was
41753 initiated, canceled, or completed.

41754 [EIO] One or more of the individual I/O operations failed. The application may
41755 check the error status for each **aiocb** structure to determine the individual
41756 request(s) that failed.

41757 In addition to the errors returned by the *lio_listio()* function, if the *lio_listio()* function succeeds
41758 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list
41759 may have been initiated. If the *lio_listio()* function fails with an error code other than [EAGAIN],
41760 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation
41761 indicated by each list element can encounter errors specific to the individual read or write
41762 function being performed. In this event, the error status for each **aiocb** control block contains the
41763 associated error code. The error codes that can be set are the same as would be set by a *read()* or
41764 *write()* function, with the following additional error codes possible:

- 41765 [EAGAIN] The requested I/O operation was not queued due to resource limitations.
- 41766 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
41767 *aio_cancel()* request.
- 41768 [EFBIG] The *aioctx* *aio_lio_opcode* is LIO_WRITE, the file is a regular file,
41769 *aioctx* *aio_nbytes* is greater than 0, and the *aioctx* *aio_offset* is greater than or
41770 equal to the offset maximum in the open file description associated with
41771 *aioctx* *aio_fildes*.
- 41772 [EINPROGRESS] The requested I/O is in progress.
- 41773 [EOVERFLOW] The *aioctx* *aio_lio_opcode* is LIO_READ, the file is a regular file,
41774 *aioctx* *aio_nbytes* is greater than 0, and the *aioctx* *aio_offset* is before the
41775 end-of-file and is greater than or equal to the offset maximum in the open file
41776 description associated with *aioctx* *aio_fildes*.

41777 EXAMPLES

41778 None.

41779 APPLICATION USAGE

41780 None.

41781 RATIONALE

41782 Although it may appear that there are inconsistencies in the specified circumstances for error
41783 codes, the [EIO] error condition applies when any circumstance relating to an individual
41784 operation makes that operation fail. This might be due to a badly formulated request (for
41785 example, the *aio_lio_opcode* field is invalid, and *aio_error()* returns [EINVAL]) or might arise from
41786 application behavior (for example, the file descriptor is closed before the operation is initiated,
41787 and *aio_error()* returns [EBADF]).

41788 The limitation on the set of error codes returned when operations from the list shall have been
41789 initiated enables applications to know when operations have been started and whether
41790 *aio_error()* is valid for a specific operation.

41791 FUTURE DIRECTIONS

41792 None.

41793 SEE ALSO

41794 *aio_read()*, *aio_write()*, *aio_error()*, *aio_return()*, *aio_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
41795 *read()*

41796 XBD <[aio.h](#)>

41797 CHANGE HISTORY

41798 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

41799 Large File Summit extensions are added.

41800 Issue 6

41801 The [ENOSYS] error condition has been removed as stubs need not be provided if an
41802 implementation does not support the Asynchronous Input and Output option.

41803 The *lio_listio()* function is marked as part of the Asynchronous Input and Output option.

41804 The following new requirements on POSIX implementations derive from alignment with the
41805 Single UNIX Specification:

41806 In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
41807 past the offset maximum established in the open file description associated with
41808 *aiocbp* ~~*aio_fildes*~~. This change is to support large files.

41809 The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support
41810 large files.

41811 The normative text is updated to avoid use of the term “must” for application requirements.

41812 The **restrict** keyword is added to the *lio_listio()* prototype for alignment with the
41813 ISO/IEC 9899:1999 standard.

41814 **Issue 6**

41815 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for
41816 symmetry with the *aio_read()* and *aio_write()* functions to the DESCRIPTION.

41817 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the
41818 DESCRIPTION making it explicit that the user is required to keep the structure pointed to by
41819 *sig* ~~*sigev_notify_attributes*~~ valid until the last asynchronous operation finished and the
41820 notification has been sent.

41821 **Issue 7**

41822 The *lio_listio()* function is moved from the Asynchronous Input and Output option to the Base.

41823 **NAME**

41824 listen ¶listen for socket connections and limit the queue of incoming connections

41825 **SYNOPSIS**

```
41826 #include <sys/socket.h>
41827 int listen(int socket, int backlog);
```

41828 **DESCRIPTION**

41829 The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as
 41830 accepting connections.

41831 The *backlog* argument provides a hint to the implementation which the implementation shall use
 41832 to limit the number of outstanding connections in the socket's listen queue. Implementations
 41833 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*
 41834 argument value shall result in a larger or equal length of the listen queue. Implementations shall
 41835 support values of *backlog* up to SOMAXCONN, defined in **<sys/socket.h>**.

41836 The implementation may include incomplete connections in its listen queue. The limits on the
 41837 number of incomplete connections and completed connections queued may be different.

41838 The implementation may have an upper limit on the length of the listen queue—either global or
 41839 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

41840 If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it
 41841 had been called with a *backlog* argument value of 0.

41842 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of
 41843 the listen queue may be set to an implementation-defined minimum value.

41844 The socket in use may require the process to have appropriate privileges to use the *listen()*
 41845 function.

41846 **RETURN VALUE**

41847 Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set
 41848 to indicate the error.

41849 **ERRORS**

41850 The *listen()* function shall fail if:

41851 [EBADF] The *socket* argument is not a valid file descriptor.

41852 [EDESTADDRREQ]

41853 The socket is not bound to a local address, and the protocol does not support
 41854 listening on an unbound socket.

41855 [EINVAL] The *socket* is already connected.

41856 [ENOTSOCK] The *socket* argument does not refer to a socket.

41857 [EOPNOTSUPP] The socket protocol does not support *listen()*.

41858 The *listen()* function may fail if:

41859 [EACCES] The calling process does not have appropriate privileges.

41860 [EINVAL] The *socket* has been shut down.

41861 [ENOBUFS] Insufficient resources are available in the system to complete the call.

41862 **EXAMPLES**

41863 None.

41864 **APPLICATION USAGE**

41865 None.

41866 **RATIONALE**

41867 None.

41868 **FUTURE DIRECTIONS**

41869 None.

41870 **SEE ALSO**41871 [accept\(\)](#), [connect\(\)](#), [socket\(\)](#)41872 XBD [<sys/socket.h>](#)41873 **CHANGE HISTORY**

41874 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

41875 The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog*
41876 argument.

41877 **NAME**

41878 llabs — return a long integer absolute value

41879 **SYNOPSIS**

41880 #include <stdlib.h>

41881 long long llabs(long long i);

41882 **DESCRIPTION**

41883 Refer to *labs()*.

41884 **NAME**

41885 lldiv — compute quotient and remainder of a long division

41886 **SYNOPSIS**

41887 #include <stdlib.h>

41888 lldiv_t lldiv(long long *numer*, long long *denom*);41889 **DESCRIPTION**41890 Refer to *ldiv()*.

41891 **NAME**

41892 llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

41893 **SYNOPSIS**

```
41894 #include <math.h>
41895 long long llrint(double x);
41896 long long llrintf(float x);
41897 long long llrintl(long double x);
```

41898 **DESCRIPTION**

41899 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41900 conflict between the requirements described here and the ISO C standard is unintentional. This
 41901 volume of POSIX.1-2017 defers to the ISO C standard.

41902 These functions shall round their argument to the nearest integer value, rounding according to
 41903 the current rounding direction.

41904 An application wishing to check for error situations should set *errno* to zero and call
 41905 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41906 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41907 zero, an error has occurred.

41908 **RETURN VALUE**

41909 Upon successful completion, these functions shall return the rounded integer value.

41910 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

41911 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

41912 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

41913 If the correct value is positive and too large to represent as a **long long**, an unspecified value
 41914 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 41915 CX shall occur; otherwise, a domain error may occur.

41916 If the correct value is negative and too large to represent as a **long long**, an unspecified value
 41917 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 41918 CX shall occur; otherwise, a domain error may occur.

41919 **ERRORS**

41920 These functions shall fail if:

41921 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 41922 integer.

41923 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41924 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41925 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41926 shall be raised.

41927 These functions may fail if:

41928 **Domain Error** The correct value is not representable as an integer.

41929 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41930 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41931 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41932 shall be raised.

41933 **EXAMPLES**

41934 None.

41935 **APPLICATION USAGE**

41936 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
41937 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

41938 **RATIONALE**

41939 These functions provide floating-to-integer conversions. They round according to the current
41940 rounding direction. If the rounded value is outside the range of the return type, the numeric
41941 result is unspecified and the invalid floating-point exception is raised. When they raise no other
41942 floating-point exception and the result differs from the argument, they raise the inexact floating-
41943 point exception.

41944 **FUTURE DIRECTIONS**

41945 None.

41946 **SEE ALSO**41947 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [lrint\(\)](#)41948 XBD [Section 4.20](#) (on page 117), [<math.h>](#)41949 **CHANGE HISTORY**

41950 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

41951 **Issue 7**

41952 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 is applied.

41953 **NAME**

41954 llround, llroundf, llroundl — round to nearest integer value

41955 **SYNOPSIS**

```
41956 #include <math.h>
41957 long long llround(double x);
41958 long long llroundf(float x);
41959 long long llroundl(long double x);
```

41960 **DESCRIPTION**

41961 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41962 conflict between the requirements described here and the ISO C standard is unintentional. This
 41963 volume of POSIX.1-2017 defers to the ISO C standard.

41964 These functions shall round their argument to the nearest integer value, rounding halfway cases
 41965 away from zero, regardless of the current rounding direction.

41966 An application wishing to check for error situations should set *errno* to zero and call
 41967 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 41968 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 41969 zero, an error has occurred.

41970 **RETURN VALUE**

41971 Upon successful completion, these functions shall return the rounded integer value.

41972 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

41973 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

41974 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

41975 If the correct value is positive and too large to represent as a **long long**, an unspecified value
 41976 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 41977 CX shall occur; otherwise, a domain error may occur.

41978 If the correct value is negative and too large to represent as a **long long**, an unspecified value
 41979 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 41980 CX shall occur; otherwise, a domain error may occur.

41981 **ERRORS**

41982 These functions shall fail if:

41983 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 41984 integer.

41985 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41986 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41987 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41988 shall be raised.

41989 These functions may fail if:

41990 **Domain Error** The correct value is not representable as an integer.

41991 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 41992 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 41993 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 41994 shall be raised.

41995 **EXAMPLES**

41996 None.

41997 **APPLICATION USAGE**41998 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
41999 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.42000 **RATIONALE**42001 These functions differ from the *llrint()* functions in that the default rounding direction for the
42002 *llround()* functions round halfway cases away from zero and need not raise the inexact floating-
42003 point exception for non-integer arguments that round to within the range of the return type.42004 **FUTURE DIRECTIONS**

42005 None.

42006 **SEE ALSO**42007 [fclearexcept\(\)](#), [fetetestexcept\(\)](#), [lround\(\)](#)42008 XBD [Section 4.20](#) (on page 117), [<math.h>](#)42009 **CHANGE HISTORY**

42010 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

42011 **Issue 7**

42012 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

42013 **NAME**

42014 localeconv — return locale-specific information

42015 **SYNOPSIS**

42016 #include <locale.h>

42017 struct lconv *localeconv(void);

42018 **DESCRIPTION**

42019 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42020 conflict between the requirements described here and the ISO C standard is unintentional. This
 42021 volume of POSIX.1-2017 defers to the ISO C standard.

42022 The *localeconv()* function shall set the components of an object with the type **struct lconv** with
 42023 the values appropriate for the formatting of numeric quantities (monetary and otherwise)
 42024 according to the rules of the current locale.

42025 The members of the structure with type **char *** are pointers to strings, any of which (except
 42026 **decimal_point**) can point to " ", to indicate that the value is not available in the current locale or
 42027 is of zero length. The members with type **char** are non-negative numbers, any of which can be
 42028 {CHAR_MAX} to indicate that the value is not available in the current locale.

42029 The members include the following:

42030 **char *decimal_point**

42031 The radix character used to format non-monetary quantities.

42032 **char *thousands_sep**

42033 The character used to separate groups of digits before the decimal-point character in
 42034 formatted non-monetary quantities.

42035 **char *grouping**

42036 A string whose elements taken as one-byte integer values indicate the size of each group of
 42037 digits in formatted non-monetary quantities.

42038 **char *int_curr_symbol**

42039 The international currency symbol applicable to the current locale. The first three characters
 42040 contain the alphabetic international currency symbol in accordance with those specified in
 42041 the ISO 4217:2001 standard. The fourth character (immediately preceding the null byte) is
 42042 the character used to separate the international currency symbol from the monetary
 42043 quantity.

42044 **char *currency_symbol**

42045 The local currency symbol applicable to the current locale.

42046 **char *mon_decimal_point**

42047 The radix character used to format monetary quantities.

42048 **char *mon_thousands_sep**

42049 The separator for groups of digits before the decimal-point in formatted monetary
 42050 quantities.

42051 **char *mon_grouping**

42052 A string whose elements taken as one-byte integer values indicate the size of each group of
 42053 digits in formatted monetary quantities.

42054 **char *positive_sign**

42055 The string used to indicate a non-negative valued formatted monetary quantity.

- 42056 **char *negative_sign**
42057 The string used to indicate a negative valued formatted monetary quantity.
- 42058 **char int_frac_digits**
42059 The number of fractional digits (those after the decimal-point) to be displayed in an
42060 internationally formatted monetary quantity.
- 42061 **char frac_digits**
42062 The number of fractional digits (those after the decimal-point) to be displayed in a
42063 formatted monetary quantity.
- 42064 **char p_cs_precedes**
42065 Set to 1 if the **currency_symbol** precedes the value for a non-negative formatted monetary
42066 quantity. Set to 0 if the symbol succeeds the value.
- 42067 **char p_sep_by_space**
42068 Set to a value indicating the separation of the **currency_symbol**, the sign string, and the
42069 value for a non-negative formatted monetary quantity.
- 42070 **char n_cs_precedes**
42071 Set to 1 if the **currency_symbol** precedes the value for a negative formatted monetary
42072 quantity. Set to 0 if the symbol succeeds the value.
- 42073 **char n_sep_by_space**
42074 Set to a value indicating the separation of the **currency_symbol**, the sign string, and the
42075 value for a negative formatted monetary quantity.
- 42076 **char p_sign_posn**
42077 Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted
42078 monetary quantity.
- 42079 **char n_sign_posn**
42080 Set to a value indicating the positioning of the **negative_sign** for a negative formatted
42081 monetary quantity.
- 42082 **char int_p_cs_precedes**
42083 Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a non-
42084 negative internationally formatted monetary quantity.
- 42085 **char int_n_cs_precedes**
42086 Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a
42087 negative internationally formatted monetary quantity.
- 42088 **char int_p_sep_by_space**
42089 Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the
42090 value for a non-negative internationally formatted monetary quantity.
- 42091 **char int_n_sep_by_space**
42092 Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the
42093 value for a negative internationally formatted monetary quantity.
- 42094 **char int_p_sign_posn**
42095 Set to a value indicating the positioning of the **positive_sign** for a non-negative
42096 internationally formatted monetary quantity.
- 42097 **char int_n_sign_posn**
42098 Set to a value indicating the positioning of the **negative_sign** for a negative internationally
42099 formatted monetary quantity.

- 42100 The elements of **grouping** and **mon_grouping** are interpreted according to the following:
- 42101 {CHAR_MAX} No further grouping is to be performed.
- 42102 0 The previous element is to be repeatedly used for the remainder of the digits.
- 42103 *other* The integer value is the number of digits that comprise the current group. The
- 42104 next element is examined to determine the size of the next group of digits
- 42105 before the current group.
- 42106 The values of **p_sep_by_space**, **n_sep_by_space**, **int_p_sep_by_space**, and **int_n_sep_by_space**
- 42107 are interpreted according to the following:
- 42108 0 No space separates the currency symbol and value.
- 42109 1 If the currency symbol and sign string are adjacent, a space separates them from the value;
- 42110 otherwise, a space separates the currency symbol from the value.
- 42111 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a
- 42112 space separates the sign string from the value.
- 42113 For **int_p_sep_by_space** and **int_n_sep_by_space**, the fourth character of **int_curr_symbol** is
- 42114 used instead of a space.
- 42115 The values of **p_sign_posn**, **n_sign_posn**, **int_p_sign_posn**, and **int_n_sign_posn** are
- 42116 interpreted according to the following:
- 42117 0 Parentheses surround the quantity and **currency_symbol** or **int_curr_symbol**.
- 42118 1 The sign string precedes the quantity and **currency_symbol** or **int_curr_symbol**.
- 42119 2 The sign string succeeds the quantity and **currency_symbol** or **int_curr_symbol**.
- 42120 3 The sign string immediately precedes the **currency_symbol** or **int_curr_symbol**.
- 42121 4 The sign string immediately succeeds the **currency_symbol** or **int_curr_symbol**.
- 42122 The implementation shall behave as if no function in this volume of POSIX.1-2017 calls
- 42123 *localeconv()*.
- 42124 CX The *localeconv()* function need not be thread-safe.
- 42125 **RETURN VALUE**
- 42126 The *localeconv()* function shall return a pointer to the filled-in object. The application shall not
- 42127 CX modify the structure to which the return value points, nor any storage areas pointed to by
- 42128 pointers within the structure. The returned pointer, and pointers within the structure, might be
- 42129 CX invalidated or the structure or the storage areas might be overwritten by a subsequent call to
- 42130 CX *localeconv()*. In addition, the returned pointer, and pointers within the structure, might be
- 42131 CX invalidated or the structure or the storage areas might be overwritten by subsequent calls to
- 42132 CX *setlocale()* with the categories LC_ALL, LC_MONETARY, or LC_NUMERIC, or by calls to
- 42133 *uselocale()* which change the categories LC_MONETARY or LC_NUMERIC. The returned
- 42134 pointer, pointers within the structure, the structure, and the storage areas might also be
- 42135 invalidated if the calling thread is terminated.
- 42136 **ERRORS**
- 42137 No errors are defined.

42138 **EXAMPLES**

42139 None.

42140 **APPLICATION USAGE**42141 The following table illustrates the rules which may be used by four countries to format
42142 monetary quantities.

Country	Positive Format	Negative Format	International Format
Italy	€1.230	-€1.230	EUR.1.230
Netherlands	€ 1.234,56	€ -1.234,56	EUR 1.234,56
Norway	kr1.234,56	kr1.234,56-	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

42148 For these four countries, the respective values for the monetary members of the structure
42149 returned by *localeconv()* are:

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"EUR. "	"EUR "	"NOK "	"CHF "
currency_symbol	"€. "	"€"	"kr"	"SFrs. "
mon_decimal_point	" "	","	","	."
mon_thousands_sep	"."	"."	"."	","
mon_grouping	"\3"	"\3"	"\3"	"\3"
positive_sign	" "	" "	" "	" "
negative_sign	"_ "	"_ "	"_ "	"C"
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

42172 **RATIONALE**

42173 None.

42174 **FUTURE DIRECTIONS**

42175 None.

42176 **SEE ALSO**42177 *fprintf()*, *fscanf()*, *isalpha()*, *isascii()*, *nl_langinfo()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*,
42178 *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, *uselocale()*42179 XBD [<langinfo.h>](#), [<locale.h>](#)

42180 **CHANGE HISTORY**

42181 First released in Issue 4. Derived from the ANSI C standard.

42182 **Issue 6**

42183 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

42184 The RETURN VALUE section is rewritten to avoid use of the term “must”.

42185 This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard.

42186 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.

42187 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These changes are for alignment with the ISO C standard.

42190 **Issue 7**

42191 Austin Group Interpretation 1003.1-2001 #156 is applied.

42192 The definitions of **int_curr_symbol** and **currency_symbol** are updated.

42193 The examples in the APPLICATION USAGE section are updated.

42194 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0362 [75] is applied.

42195 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0200 [656] is applied.

42196 **NAME**

42197 localtime, localtime_r — convert a time value to a broken-down local time

42198 **SYNOPSIS**

42199 #include <time.h>

42200 struct tm *localtime(const time_t *timer);

42201 CX struct tm *localtime_r(const time_t *restrict timer,
42202 struct tm *restrict result);42203 **DESCRIPTION**42204 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C
42205 standard. Any conflict between the requirements described here and the ISO C standard is
42206 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.42207 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer*
42208 into a broken-down time, expressed as a local time. The function corrects for the timezone and
42209 CX any seasonal time adjustments. Local timezone information is used as though *localtime()* calls
42210 *tzset()*.42211 The relationship between a time in seconds since the Epoch used as an argument to *localtime()*
42212 and the **tm** structure (defined in the <**time.h**> header) is that the result shall be as specified in
42213 the expression given in the definition of seconds since the Epoch (see XBD [Section 4.16](#), on page
42214 113) corrected for timezone and any seasonal time adjustments, where the names in the structure
42215 and in the expression correspond.42216 The same relationship shall apply for *localtime_r()*.42217 The *localtime()* function need not be thread-safe.42218 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
42219 objects: a broken-down time structure and an array of type **char**. Execution of any of the
42220 functions may overwrite the information returned in either of these objects by any of the other
42221 functions.42222 The *localtime_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*
42223 into a broken-down time stored in the structure to which *result* points. The *localtime_r()* function
42224 shall also return a pointer to that same structure.42225 Unlike *localtime()*, the *localtime_r()* function is not required to set *tzname*. If *localtime_r()* sets
42226 *tzname*, it shall also set *daylight* and *timezone*. If *localtime_r()* does not set *tzname*, it shall not set
42227 *daylight* and shall not set *timezone*.42228 **RETURN VALUE**42229 Upon successful completion, the *localtime()* function shall return a pointer to the broken-down
42230 CX time structure. If an error is detected, *localtime()* shall return a null pointer and set *errno* to
42231 indicate the error.42232 Upon successful completion, *localtime_r()* shall return a pointer to the structure pointed to by the
42233 argument *result*. If an error is detected, *localtime_r()* shall return a null pointer and set *errno* to
42234 indicate the error.42235 **ERRORS**42236 CX The *localtime()* and *localtime_r()* functions shall fail if:42237 CX [E`OVERFLOW`] The result cannot be represented.

42238 **EXAMPLES**42239 **Getting the Local Date and Time**

42240 The following example uses the *time()* function to calculate the time elapsed, in seconds, since
 42241 January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time,
 42242 and *asctime()* to convert the broken-down time values into a printable string.

```
42243 #include <stdio.h>
42244 #include <time.h>
42245 int main(void)
42246 {
42247     time_t result;
42248
42249     result = time(NULL);
42250     printf("%s%ju secs since the Epoch\n",
42251           asctime(localtime(&result)),
42252           (uintmax_t)result);
42253     return(0);
42254 }
```

42254 This example writes the current time to *stdout* in a form like this:

```
42255 Wed Jun 26 10:32:15 1996
42256 835810335 secs since the Epoch
```

42257 **Getting the Modification Time for a File**

42258 The following example prints the last data modification timestamp in the local timezone for a
 42259 given file.

```
42260 #include <stdio.h>
42261 #include <time.h>
42262 #include <sys/stat.h>
42263 int
42264 print_file_time(const char *pathname)
42265 {
42266     struct stat statbuf;
42267     struct tm *tm;
42268     char timestr[BUFSIZ];
42269
42270     if(stat(pathname, &statbuf) == -1)
42271         return -1;
42272     if((tm = localtime(&statbuf.st_mtime)) == NULL)
42273         return -1;
42274     if(strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
42275         return -1;
42276     printf("%s: %s.%09ld\n", pathname, timestr, statbuf.st_mtim.tv_nsec);
42277     return 0;
42278 }
```

42278 **Timing an Event**

42279 The following example gets the current time, converts it to a string using *localtime()* and
 42280 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
 42281 an event being timed.

```
42282 #include <time.h>
42283 #include <stdio.h>
42284 ...
42285 time_t now;
42286 int minutes_to_event;
42287 ...
42288 time(&now);
42289 printf("The time is ");
42290 fputs(asctime(localtime(&now)), stdout);
42291 printf("There are still %d minutes to the event.\n",
42292     minutes_to_event);
42293 ...
```

42294 **APPLICATION USAGE**

42295 The *localtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 42296 possibly using a static data area that may be overwritten by each call.

42297 **RATIONALE**

42298 None.

42299 **FUTURE DIRECTIONS**

42300 None.

42301 **SEE ALSO**

42302 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time()*,
 42303 *tzset()*, *utime()*

42304 XBD Section 4.16 (on page 113), [<time.h>](#)

42305 **CHANGE HISTORY**

42306 First released in Issue 1. Derived from Issue 1 of the SVID.

42307 **Issue 5**

42308 A note indicating that the *localtime()* function need not be reentrant is added to the
 42309 DESCRIPTION.

42310 The *localtime_r()* function is included for alignment with the POSIX Threads Extension.

42311 **Issue 6**

42312 The *localtime_r()* function is marked as part of the Thread-Safe Functions option.

42313 Extensions beyond the ISO C standard are marked.

42314 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 42315 its avoidance of possibly using a static data area.

42316 The **restrict** keyword is added to the *localtime_r()* prototype for alignment with the
 42317 ISO/IEC 9899:1999 standard.

42318 Examples are added.

42319 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the
 42320 [EOverflow] error.

- 42321 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/55 is applied, updating the error handling
42322 for *localtime_r()*.
- 42323 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/56 is applied, adding a requirement that if
42324 *localtime_r()* does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On
42325 systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide
42326 information for the same timezone. This updates the description of *localtime_r()* to mention
42327 *daylight* and *timezone* as well as *tzname*. The SEE ALSO section is updated.
- 42328 **Issue 7**
- 42329 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 42330 The *localtime_r()* function is moved from the Thread-Safe Functions option to the Base.
- 42331 Changes are made to the EXAMPLES section related to support for finegrained timestamps.
- 42332 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0363 [291] is applied.
- 42333 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0201 [664] is applied.

42334 **NAME**

42335 lockf — record locking on files

42336 **SYNOPSIS**

```
42337 XSI #include <unistd.h>
42338 int lockf(int fildev, int function, off_t size);
```

42339 **DESCRIPTION**

42340 The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from
 42341 threads in other processes which attempt to lock the locked file section shall either return an
 42342 error value or block until the section becomes unlocked. All the locks for a process are removed
 42343 when the process terminates. Record locking with *lockf()* shall be supported for regular files and
 42344 may be supported for other files.

42345 The *fildev* argument is an open file descriptor. To establish a lock with this function, the file
 42346 descriptor shall be opened with write-only permission (O_WRONLY) or with read/write
 42347 permission (O_RDWR).

42348 The *function* argument is a control value which specifies the action to be taken. The permissible
 42349 values for *function* are defined in <unistd.h> as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

42350 F_TEST shall detect if a lock by another process is present on the specified section.

42351 F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

42352 F_ULOCK shall remove locks from a section of the file.

42353 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be
 42354 locked or unlocked starts at the current offset in the file and extends forward for a positive size
 42355 or backward for a negative size (the preceding bytes up to but not including the current offset).
 42356 If *size* is 0, the section from the current offset through the largest possible file offset shall be
 42357 locked (that is, from the current offset through the present or any future end-of-file). An area
 42358 need not be allocated to the file to be locked because locks may exist past the end-of-file.

42359 The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained
 42360 by a previously locked section for the same process. When this occurs, or if adjacent locked
 42361 sections would occur, the sections shall be combined into a single locked section. If the request
 42362 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

42363 F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available.
 42364 F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the
 42365 function to fail if the section is already locked by another process.

42366 File locks shall be released 42367 first close by the locking process of any file descriptor for the file.

42368 F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the
 42369 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or
 42370 to the end-of-file if *size* is (off_t)0. When all of a locked section is not released (that is, when the
 42371 beginning or end of the area to be unlocked falls within a locked section), the remaining portions
 42372 of that section shall remain locked by the process. Releasing the center portion of a locked
 42373 section shall cause the remaining locked beginning and end portions to become two separate

42378 locked sections. If the request would cause the number of locks in the system to exceed a system-
42379 imposed limit, the request shall fail.

42380 A potential for deadlock occurs if the threads of a process controlling a locked section are
42381 blocked by accessing a locked section of another process. If the system detects that deadlock
42382 would occur, *lockf()* shall fail with an [EDEADLK] error.

42383 The interaction between *fcntl()* and *lockf()* locks is unspecified.

42384 Blocking on a section shall be interrupted by any signal.

42385 An F_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested
42386 section is the maximum value for an object of type **off_t**, when the process has an existing lock
42387 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a
42388 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an
42389 F_ULOCK request shall attempt to unlock only the requested section.

42390 Attempting to lock a section of a file that is associated with a buffered stream produces
42391 unspecified results.

42392 RETURN VALUE

42393 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to
42394 indicate an error, and existing locks shall not be changed.

42395 ERRORS

42396 The *lockf()* function shall fail if:

42397 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F_LOCK
42398 or F_TLOCK and *fildev* is not a valid file descriptor open for writing.

42399 [EACCES] or [EAGAIN]

42400 The *function* argument is F_TLOCK or F_TEST and the section is already
42401 locked by another process.

42402 [EDEADLK] The *function* argument is F_LOCK and a deadlock is detected.

42403 [EINTR] A signal was caught during execution of the function.

42404 [EINVAL] The *function* argument is not one of F_LOCK, F_TLOCK, F_TEST, or
42405 F_ULOCK; or *size* plus the current file offset is less than 0.

42406 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested
42407 section cannot be represented correctly in an object of type **off_t**.

42408 The *lockf()* function may fail if:

42409 [EAGAIN] The *function* argument is F_LOCK or F_TLOCK and the file is mapped with
42410 *mmap()*.

42411 [EDEADLK] or [ENOLCK]

42412 The *function* argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request
42413 would cause the number of locks to exceed a system-imposed limit.

42414 [EOPNOTSUPP] or [EINVAL]

42415 The implementation does not support the locking of files of the type indicated
42416 by the *fildev* argument.

42417 **EXAMPLES**42418 **Locking a Portion of a File**

42419 In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that
 42420 use locking are prevented from changing it during this process. Only the first 10 000 bytes are
 42421 locked, and the lock call fails if another process has any part of this area locked already.

```
42422 #include <fcntl.h>
42423 #include <unistd.h>
42424 int fildes;
42425 int status;
42426 ...
42427 fildes = open("/home/cnd/mod1", O_RDWR);
42428 status = lockf(fildes, F_TLOCK, (off_t)10000);
```

42429 **APPLICATION USAGE**

42430 Record-locking should not be used in combination with the `fopen()`, `fread()`, `fwrite()`, and other
 42431 `stdio` functions. Instead, the more primitive, non-buffered functions (such as `open()`) should be
 42432 used. Unexpected results may occur in processes that do buffering in the user address space. The
 42433 process may later read/write data which is/was locked. The `stdio` functions are the most
 42434 common source of unexpected buffering.

42435 The `alarm()` function may be used to provide a timeout facility in applications requiring it.

42436 **RATIONALE**

42437 None.

42438 **FUTURE DIRECTIONS**

42439 None.

42440 **SEE ALSO**

42441 [alarm\(\)](#), [chmod\(\)](#), [close\(\)](#), [creat\(\)](#), [fcntl\(\)](#), [fopen\(\)](#), [mmap\(\)](#), [open\(\)](#), [read\(\)](#), [write\(\)](#)

42442 XBD [<unistd.h>](#)

42443 **CHANGE HISTORY**

42444 First released in Issue 4, Version 2.

42445 **Issue 5**

42446 Moved from X/OPEN UNIX extension to BASE.

42447 Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified
 42448 and moved from optional to mandatory status.

42449 A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a
 42450 file that is associated with a buffered stream.

42451 **Issue 6**

42452 The normative text is updated to avoid use of the term “must” for application requirements.

42453 **Issue 7**

42454 Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION.

42455 **NAME**

42456 log, logf, logl ‡natural logarithm function

42457 **SYNOPSIS**

```
42458 #include <math.h>
42459 double log(double x);
42460 float logf(float x);
42461 long double logl(long double x);
```

42462 **DESCRIPTION**

42463 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42464 conflict between the requirements described here and the ISO C standard is unintentional. This
 42465 volume of POSIX.1-2017 defers to the ISO C standard.

42466 These functions shall compute the natural logarithm of their argument x , $\log_e(x)$.

42467 An application wishing to check for error situations should set *errno* to zero and call
 42468 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42469 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42470 zero, an error has occurred.

42471 **RETURN VALUE**

42472 Upon successful completion, these functions shall return the natural logarithm of x .

42473 If x is ± 0 , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return $-\text{HUGE_VAL}$,
 42474 $-\text{HUGE_VALF}$, and $-\text{HUGE_VALL}$, respectively.

42475 MX For finite values of x that are less than 0, or if x is $-\text{Inf}$, a domain error shall occur, and either a
 42476 NaN (if supported), or an implementation-defined value shall be returned.

42477 MX If x is NaN, a NaN shall be returned.

42478 If x is 1, +0 shall be returned.

42479 If x is +Inf, x shall be returned.

42480 **ERRORS**

42481 These functions shall fail if:

42482 MX Domain Error The finite value of x is negative, or x is $-\text{Inf}$.

42483 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42484 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42485 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42486 shall be raised.

42487 Pole Error The value of x is zero.

42488 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42489 then *errno* shall be set to [ERANGE]. If the integer expression
 42490 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 42491 floating-point exception shall be raised.

42492 EXAMPLES

42493 None.

42494 APPLICATION USAGE

42495 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
42496 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42497 RATIONALE

42498 None.

42499 FUTURE DIRECTIONS

42500 None.

42501 SEE ALSO

42502 [exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log10\(\)](#), [log1p\(\)](#)

42503 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

42504 CHANGE HISTORY

42505 First released in Issue 1. Derived from Issue 1 of the SVID.

42506 Issue 5

42507 The DESCRIPTION is updated to indicate how an application should check for an error. This
42508 text was previously published in the APPLICATION USAGE section.

42509 Issue 6

42510 The normative text is updated to avoid use of the term “must” for application requirements.

42511 The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

42512 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
42513 revised to align with the ISO/IEC 9899:1999 standard.

42514 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
42515 marked.

42516 **NAME**42517 `log10, log10f, log10l` ‡base 10 logarithm function42518 **SYNOPSIS**

```
42519 #include <math.h>
42520 double log10(double x);
42521 float log10f(float x);
42522 long double log10l(long double x);
```

42523 **DESCRIPTION**

42524 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42525 conflict between the requirements described here and the ISO C standard is unintentional. This
 42526 volume of POSIX.1-2017 defers to the ISO C standard.

42527 These functions shall compute the base 10 logarithm of their argument x , $\log_{10}(x)$.

42528 An application wishing to check for error situations should set *errno* to zero and call
 42529 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42530 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42531 zero, an error has occurred.

42532 **RETURN VALUE**

42533 Upon successful completion, these functions shall return the base 10 logarithm of x .

42534 If x is ± 0 , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return `-HUGE_VAL`,
 42535 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

42536 MX For finite values of x that are less than 0, or if x is `-Inf`, a domain error shall occur, and either a
 42537 NaN (if supported), or an implementation-defined value shall be returned.

42538 MX If x is NaN, a NaN shall be returned.

42539 If x is 1, `+0` shall be returned.

42540 If x is `+Inf`, `+Inf` shall be returned.

42541 **ERRORS**

42542 These functions shall fail if:

42543 MX Domain Error The finite value of x is negative, or x is `-Inf`.

42544 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42545 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42546 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42547 shall be raised.

42548 Pole Error The value of x is zero.

42549 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42550 then *errno* shall be set to [ERANGE]. If the integer expression
 42551 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 42552 floating-point exception shall be raised.

42553 **EXAMPLES**

42554 None.

42555 **APPLICATION USAGE**

42556 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
42557 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42558 **RATIONALE**

42559 None.

42560 **FUTURE DIRECTIONS**

42561 None.

42562 **SEE ALSO**42563 *feclearexcept()*, *fetestexcept()*, *isnan()*, *log()*, *pow()*

42564 XBD Section 4.20 (on page 117), <math.h>

42565 **CHANGE HISTORY**

42566 First released in Issue 1. Derived from Issue 1 of the SVID.

42567 **Issue 5**

42568 The DESCRIPTION is updated to indicate how an application should check for an error. This
42569 text was previously published in the APPLICATION USAGE section.

42570 **Issue 6**

42571 The normative text is updated to avoid use of the term “must” for application requirements.

42572 The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999
42573 standard.

42574 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
42575 revised to align with the ISO/IEC 9899:1999 standard.

42576 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
42577 marked.

42578 **NAME**42579 `log1p`, `log1pf`, `log1pl` ‡compute a natural logarithm42580 **SYNOPSIS**

```
42581 #include <math.h>
42582 double log1p(double x);
42583 float log1pf(float x);
42584 long double log1pl(long double x);
```

42585 **DESCRIPTION**

42586 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42587 conflict between the requirements described here and the ISO C standard is unintentional. This
 42588 volume of POSIX.1-2017 defers to the ISO C standard.

42589 These functions shall compute $\log_e(1.0 + x)$.

42590 An application wishing to check for error situations should set *errno* to zero and call
 42591 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42592 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42593 zero, an error has occurred.

42594 **RETURN VALUE**

42595 Upon successful completion, these functions shall return the natural logarithm of $1.0 + x$.

42596 If x is -1 , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return `-HUGE_VAL`,
 42597 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

42598 MX For finite values of x that are less than -1 , or if x is $-\text{Inf}$, a domain error shall occur, and either a
 42599 NaN (if supported), or an implementation-defined value shall be returned.

42600 MX If x is NaN, a NaN shall be returned.

42601 If x is ± 0 , or $+\text{Inf}$, x shall be returned.

42602 If x is subnormal, a range error may occur

42603 MXX and x should be returned.

42604 MX If x is not returned, *log1p()*, *log1pf()*, and *log1pl()* shall return an implementation-defined value
 42605 no greater in magnitude than `DBL_MIN`, `FLT_MIN`, and `LDBL_MIN`, respectively.

42606 **ERRORS**

42607 These functions shall fail if:

42608 MX Domain Error The finite value of x is less than -1 , or x is $-\text{Inf}$.

42609 If the integer expression (*math_errhandling* & `MATH_ERRNO`) is non-zero,
 42610 then *errno* shall be set to `[EDOM]`. If the integer expression (*math_errhandling*
 42611 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception
 42612 shall be raised.

42613 Pole Error The value of x is -1 .

42614 If the integer expression (*math_errhandling* & `MATH_ERRNO`) is non-zero,
 42615 then *errno* shall be set to `[ERANGE]`. If the integer expression
 42616 (*math_errhandling* & `MATH_ERREXCEPT`) is non-zero, then the divide-by-zero
 42617 floating-point exception shall be raised.

42618 These functions may fail if:

42619 MX **Range Error** The value of x is subnormal.

42620 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42621 then *errno* shall be set to [ERANGE]. If the integer expression
 42622 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 42623 floating-point exception shall be raised.

42624 **EXAMPLES**

42625 None.

42626 **APPLICATION USAGE**

42627 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 42628 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42629 **RATIONALE**

42630 None.

42631 **FUTURE DIRECTIONS**

42632 None.

42633 **SEE ALSO**

42634 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [log\(\)](#)

42635 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

42636 **CHANGE HISTORY**

42637 First released in Issue 4, Version 2.

42638 **Issue 5**

42639 Moved from X/OPEN UNIX extension to BASE.

42640 **Issue 6**

42641 The normative text is updated to avoid use of the term “must” for application requirements.

42642 The *log1p()* function is no longer marked as an extension.

42643 The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999
 42644 standard.

42645 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 42646 revised to align with the ISO/IEC 9899:1999 standard.

42647 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 42648 marked.

42649 **Issue 7**

42650 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0364 [68] is applied.

42651 **NAME**42652 `log2, log2f, log2l` ‡compute base 2 logarithm functions42653 **SYNOPSIS**

```
42654 #include <math.h>
42655 double log2(double x);
42656 float log2f(float x);
42657 long double log2l(long double x);
```

42658 **DESCRIPTION**

42659 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42660 conflict between the requirements described here and the ISO C standard is unintentional. This
 42661 volume of POSIX.1-2017 defers to the ISO C standard.

42662 These functions shall compute the base 2 logarithm of their argument x , $\log_2(x)$.

42663 An application wishing to check for error situations should set *errno* to zero and call
 42664 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42665 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42666 zero, an error has occurred.

42667 **RETURN VALUE**

42668 Upon successful completion, these functions shall return the base 2 logarithm of x .

42669 If x is ± 0 , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return `-HUGE_VAL`,
 42670 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

42671 MX For finite values of x that are less than 0, or if x is `-Inf`, a domain error shall occur, and either a
 42672 NaN (if supported), or an implementation-defined value shall be returned.

42673 MX If x is NaN, a NaN shall be returned.

42674 If x is 1, `+0` shall be returned.

42675 If x is `+Inf`, x shall be returned.

42676 **ERRORS**

42677 These functions shall fail if:

42678 MX Domain Error The finite value of x is less than zero, or x is `-Inf`.

42679 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42680 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42681 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42682 shall be raised.

42683 Pole Error The value of x is zero.

42684 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42685 then *errno* shall be set to [ERANGE]. If the integer expression
 42686 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 42687 floating-point exception shall be raised.

42688 **EXAMPLES**

42689 None.

42690 **APPLICATION USAGE**

42691 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
42692 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42693 **RATIONALE**

42694 None.

42695 **FUTURE DIRECTIONS**

42696 None.

42697 **SEE ALSO**42698 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [log\(\)](#)42699 XBD [Section 4.20](#) (on page 117), [<math.h>](#)42700 **CHANGE HISTORY**

42701 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

42702 **NAME**42703 logb, logbf, logbl \dagger radix-independent exponent42704 **SYNOPSIS**

```
42705 #include <math.h>
42706 double logb(double x);
42707 float logbf(float x);
42708 long double logbl(long double x);
```

42709 **DESCRIPTION**

42710 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42711 conflict between the requirements described here and the ISO C standard is unintentional. This
 42712 volume of POSIX.1-2017 defers to the ISO C standard.

42713 These functions shall compute the exponent of x , which is the integral part of $\log_r |x|$, as a
 42714 signed floating-point value, for non-zero x , where r is the radix of the machine's floating-point
 42715 arithmetic, which is the value of FLT_RADIX defined in the **<float.h>** header.

42716 If x is subnormal it is treated as though it were normalized; thus for finite positive x :

42717 $1 \leq x * \text{FLT_RADIX}^{-\text{logb}(x)} < \text{FLT_RADIX}$

42718 An application wishing to check for error situations should set *errno* to zero and call
 42719 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42720 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42721 zero, an error has occurred.

42722 **RETURN VALUE**

42723 Upon successful completion, these functions shall return the exponent of x .

42724 If x is ± 0 , *logb()*, *logbf()*, and *logbl()* shall return $-\text{HUGE_VAL}$, $-\text{HUGE_VALF}$, and
 42725 $-\text{HUGE_VALL}$, respectively.

42726 MX On systems that support the IEC 60559 Floating-Point option, a pole error shall occur;

42727 CX otherwise, a pole error may occur.

42728 MX If x is NaN, a NaN shall be returned.

42729 MX If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

42730 **ERRORS**

42731 These functions shall fail if:

42732 MX Pole Error The value of x is ± 0 .

42733 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42734 then *errno* shall be set to [ERANGE]. If the integer expression
 42735 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 42736 floating-point exception shall be raised.

42737 These functions may fail if:

42738 Pole Error The value of x is 0.

42739 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42740 then *errno* shall be set to [ERANGE]. If the integer expression
 42741 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 42742 floating-point exception shall be raised.

42743 **EXAMPLES**

42744 None.

42745 **APPLICATION USAGE**42746 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
42747 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.42748 **RATIONALE**

42749 None.

42750 **FUTURE DIRECTIONS**

42751 None.

42752 **SEE ALSO**42753 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ilogb\(\)](#), [scalbln\(\)](#)42754 XBD Section 4.20 (on page 117), [<float.h>](#), [<math.h>](#)42755 **CHANGE HISTORY**

42756 First released in Issue 4, Version 2.

42757 **Issue 5**

42758 Moved from X/OPEN UNIX extension to BASE.

42759 **Issue 6**42760 The *logb()* function is no longer marked as an extension.42761 The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.42762 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
42763 revised to align with the ISO/IEC 9899:1999 standard.42764 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
42765 marked.42766 **Issue 7**

42767 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

42768 **NAME**

42769 logf, logl ‡natural logarithm function

42770 **SYNOPSIS**

42771 #include <math.h>

42772 float logf(float x);

42773 long double logl(long double x);

42774 **DESCRIPTION**

42775 Refer to *log()*.

42776 **NAME**

42777 longjmp ‡non-local goto

42778 **SYNOPSIS**

42779 #include <setjmp.h>

42780 void longjmp(jmp_buf env, int val);

42781 **DESCRIPTION**

42782 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42783 conflict between the requirements described here and the ISO C standard is unintentional. This
 42784 volume of POSIX.1-2017 defers to the ISO C standard.

42785 The *longjmp()* function shall restore the environment saved by the most recent invocation of
 42786 *setjmp()* in the same process, with the corresponding **jmp_buf** argument. If the most recent
 42787 invocation of *setjmp()* with the corresponding **jmp_buf** occurred in another thread, or if there is
 42788 no such invocation, or if the function containing the invocation of *setjmp()* has terminated
 42789 execution in the interim, or if the invocation of *setjmp()* was within the scope of an identifier
 42790 with variably modified type and execution has left that scope in the interim, the behavior is
 42791 CX undefined. It is unspecified whether *longjmp()* restores the signal mask, leaves the signal mask
 42792 unchanged, or restores it to its value at the time *setjmp()* was called.

42793 All accessible objects have values, and all other components of the abstract machine have state
 42794 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,
 42795 except that the values of objects of automatic storage duration are unspecified if they meet all
 42796 the following conditions:

42797 They are local to the function containing the corresponding *setjmp()* invocation.

42798 They do not have volatile-qualified type.

42799 They are changed between the *setjmp()* invocation and *longjmp()* call.

42800 CX Although *longjmp()* is an async-signal-safe function, if it is invoked from a signal handler which
 42801 interrupted a non-async-signal-safe function or equivalent (such as the processing equivalent to
 42802 *exit()* performed after a return from the initial call to *main()*), the behavior of any subsequent
 42803 call to a non-async-signal-safe function or equivalent is undefined.

42804 The effect of a call to *longjmp()* where initialization of the **jmp_buf** structure was not performed
 42805 in the calling thread is undefined.

42806 **RETURN VALUE**

42807 After *longjmp()* is completed, program execution continues as if the corresponding invocation of
 42808 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause
 42809 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

42810 **ERRORS**

42811 No errors are defined.

42812 **EXAMPLES**

42813 None.

42814 **APPLICATION USAGE**

42815 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*
 42816 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the
 42817 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under
 42818 application control).

42819 It is recommended that applications do not call *longjmp()* or *siglongjmp()* from signal handlers.
 42820 To avoid undefined behavior when calling these functions from a signal handler, the application

- 42821 needs to ensure one of the following two things:
- 42822 1. After the call to *longjmp()* or *siglongjmp()* the process only calls async-signal-safe
42823 functions and does not return from the initial call to *main()*.
 - 42824 2. Any signal whose handler calls *longjmp()* or *siglongjmp()* is blocked during *every* call to a
42825 non-async-signal-safe function, and no such calls are made after returning from the initial
42826 call to *main()*.

42827 RATIONALE

42828 None.

42829 FUTURE DIRECTIONS

42830 None.

42831 SEE ALSO

42832 *setjmp()*, *sigaction()*, *siglongjmp()*, *sigsetjmp()*

42833 XBD <[setjmp.h](#)>

42834 CHANGE HISTORY

42835 First released in Issue 1. Derived from Issue 1 of the SVID.

42836 Issue 5

42837 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

42838 Issue 6

42839 Extensions beyond the ISO C standard are marked.

42840 The following new requirements on POSIX implementations derive from alignment with the
42841 Single UNIX Specification:

42842 The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask
42843 unspecified.

42844 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

42845 Issue 7

42846 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0365 [394] is applied.

42847 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0202 [516] is applied.

42848 **NAME**

42849 lrand48 †generate uniformly distributed pseudo-random non-negative long integers

42850 **SYNOPSIS**

42851 XSI #include <stdlib.h>

42852 long lrand48(void);

42853 **DESCRIPTION**42854 Refer to *drand48()*.

42855 **NAME**

42856 lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

42857 **SYNOPSIS**

```
42858 #include <math.h>
42859 long lrint(double x);
42860 long lrintf(float x);
42861 long lrintl(long double x);
```

42862 **DESCRIPTION**

42863 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42864 conflict between the requirements described here and the ISO C standard is unintentional. This
 42865 volume of POSIX.1-2017 defers to the ISO C standard.

42866 These functions shall round their argument to the nearest integer value, rounding according to
 42867 the current rounding direction.

42868 An application wishing to check for error situations should set *errno* to zero and call
 42869 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42870 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42871 zero, an error has occurred.

42872 **RETURN VALUE**

42873 Upon successful completion, these functions shall return the rounded integer value.

42874 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.42875 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.42876 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

42877 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be
 42878 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall
 42879 CX occur; otherwise, a domain error may occur.

42880 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be
 42881 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall
 42882 CX occur; otherwise, a domain error may occur.

42883 **ERRORS**

42884 These functions shall fail if:

42885 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 42886 integer.

42887 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42888 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42889 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42890 shall be raised.

42891 These functions may fail if:

42892 **Domain Error** The correct value is not representable as an integer.

42893 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42894 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42895 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42896 shall be raised.

42897 **EXAMPLES**

42898 None.

42899 **APPLICATION USAGE**42900 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
42901 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.42902 **RATIONALE**42903 These functions provide floating-to-integer conversions. They round according to the current
42904 rounding direction. If the rounded value is outside the range of the return type, the numeric
42905 result is unspecified and the invalid floating-point exception is raised. When they raise no other
42906 floating-point exception and the result differs from the argument, they raise the inexact floating-
42907 point exception.42908 **FUTURE DIRECTIONS**

42909 None.

42910 **SEE ALSO**42911 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [llrint\(\)](#)42912 XBD [Section 4.20](#) (on page 117), [<math.h>](#)42913 **CHANGE HISTORY**

42914 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

42915 **Issue 7**

42916 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

42917 **NAME**

42918 lround, lroundf, lroundl — round to nearest integer value

42919 **SYNOPSIS**

```
42920 #include <math.h>
42921 long lround(double x);
42922 long lroundf(float x);
42923 long lroundl(long double x);
```

42924 **DESCRIPTION**

42925 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42926 conflict between the requirements described here and the ISO C standard is unintentional. This
 42927 volume of POSIX.1-2017 defers to the ISO C standard.

42928 These functions shall round their argument to the nearest integer value, rounding halfway cases
 42929 away from zero, regardless of the current rounding direction.

42930 An application wishing to check for error situations should set *errno* to zero and call
 42931 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 42932 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 42933 zero, an error has occurred.

42934 **RETURN VALUE**

42935 Upon successful completion, these functions shall return the rounded integer value.

42936 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

42937 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

42938 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

42939 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be
 42940 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;
 42941 CX otherwise, a domain error may occur.

42942 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be
 42943 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;
 42944 CX otherwise, a domain error may occur.

42945 **ERRORS**

42946 These functions shall fail if:

42947 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 42948 integer.

42949 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42950 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42951 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42952 shall be raised.

42953 These functions may fail if:

42954 **Domain Error** The correct value is not representable as an integer.
 42955 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 42956 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 42957 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 42958 shall be raised.

42959 **EXAMPLES**

42960 None.

42961 **APPLICATION USAGE**

42962 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
42963 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

42964 **RATIONALE**

42965 These functions differ from the *lrint()* functions in the default rounding direction, with the
42966 *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact
42967 floating-point exception for non-integer arguments that round to within the range of the return
42968 type.

42969 **FUTURE DIRECTIONS**

42970 None.

42971 **SEE ALSO**42972 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [llround\(\)](#)42973 XBD [Section 4.20](#) (on page 117), [<math.h>](#)42974 **CHANGE HISTORY**

42975 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

42976 **Issue 7**

42977 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.

42978 **NAME**

42979 lsearch, lfind — linear search and update

42980 **SYNOPSIS**

```

42981 XSI #include <search.h>
42982 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
42983             int (*compar)(const void *, const void *));
42984 void *lfind(const void *key, const void *base, size_t *nel,
42985            size_t width, int (*compar)(const void *, const void *));

```

42986 **DESCRIPTION**

42987 The *lsearch()* function shall linearly search the table and return a pointer into the table for the
 42988 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*
 42989 argument points to the entry to be sought in the table. The *base* argument points to the first
 42990 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument
 42991 points to an integer containing the current number of elements in the table. The integer to which
 42992 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to
 42993 a comparison function which the application shall supply (for example, *strcmp()*). It is called
 42994 with two arguments that point to the elements being compared. The application shall ensure
 42995 that the function returns 0 if the elements are equal, and non-zero otherwise.

42996 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not
 42997 added to the table. Instead, a null pointer is returned.

42998 **RETURN VALUE**

42999 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it.
 43000 Otherwise, *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly
 43001 added element.

43002 Both functions shall return a null pointer in case of error.

43003 **ERRORS**

43004 No errors are defined.

43005 **EXAMPLES**43006 **Storing Strings in a Table**

43007 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to
 43008 ELSIZE and stores them in a table, eliminating duplicates.

```

43009 #include <stdio.h>
43010 #include <string.h>
43011 #include <search.h>
43012 #define TABSIZE 50
43013 #define ELSIZE 120
43014 ...
43015     char line[ELSIZE], tab[TABSIZE][ELSIZE];
43016     size_t nel = 0;
43017     ...
43018     while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
43019         (void) lsearch(line, tab, &nel,
43020                      ELSIZE, (int (*)(const void *, const void *)) strcmp);
43021     ...

```

43022 Finding a Matching Entry

43023 The following example finds any line that reads "This is a test.".

```
43024 #include <search.h>
43025 #include <string.h>
43026 ...
43027 char line[ELSIZE], tab[TABSIZE][ELSIZE];
43028 size_t nel = 0;
43029 char *findline;
43030 void *entry;

43031 findline = "This is a test.\n";

43032 entry = lfind(findline, tab, &nel, ELSIZE, (
43033     int (*)(const void *, const void *)) strcmp);
```

43034 APPLICATION USAGE

43035 The comparison function need not compare every byte, so arbitrary data may be contained in
43036 the elements in addition to the values being compared.

43037 Undefined results can occur if there is not enough room in the table to add a new item.

43038 RATIONALE

43039 None.

43040 FUTURE DIRECTIONS

43041 None.

43042 SEE ALSO

43043 *hcreate(), tdelete()*

43044 XBD [<search.h>](#)

43045 CHANGE HISTORY

43046 First released in Issue 1. Derived from Issue 1 of the SVID.

43047 Issue 6

43048 The normative text is updated to avoid use of the term “must” for application requirements.

43049 **NAME**

43050 lseek — move the read/write file offset

43051 **SYNOPSIS**

43052 #include <unistd.h>

43053 off_t lseek(int *fildev*, off_t *offset*, int *whence*);43054 **DESCRIPTION**43055 The *lseek()* function shall set the file offset for the open file description associated with the file
43056 descriptor *fildev*, as follows:43057 If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.43058 If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.43059 If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.

43060 The symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined in <unistd.h>.

43061 The behavior of *lseek()* on devices which are incapable of seeking is implementation-defined.
43062 The value of the file offset associated with such a device is undefined.43063 The *lseek()* function shall allow the file offset to be set beyond the end of the existing data in the
43064 file. If data is later written at this point, subsequent reads of data in the gap shall return bytes
43065 with the value 0 until data is actually written into the gap.43066 The *lseek()* function shall not, by itself, extend the size of a file.43067 SHM If *fildev* refers to a shared memory object, the result of the *lseek()* function is unspecified.43068 TYM If *fildev* refers to a typed memory object, the result of the *lseek()* function is unspecified.43069 **RETURN VALUE**43070 Upon successful completion, the resulting offset, as measured in bytes from the beginning of the
43071 file, shall be returned. Otherwise, -1 shall be returned, *errno* shall be set to indicate the error, and
43072 the file offset shall remain unchanged.43073 **ERRORS**43074 The *lseek()* function shall fail if:43075 [EBADF] The *fildev* argument is not an open file descriptor.43076 [EINVAL] The *whence* argument is not a proper value, or the resulting file offset would
43077 be negative for a regular file, block special file, or directory.43078 [EOVERFLOW] The resulting file offset would be a value which cannot be represented
43079 correctly in an object of type **off_t**.43080 [ESPIPE] The *fildev* argument is associated with a pipe, FIFO, or socket.43081 **EXAMPLES**

43082 None.

43083 **APPLICATION USAGE**

43084 None.

43085 **RATIONALE**43086 The ISO C standard includes the functions *fgetpos()* and *fsetpos()*, which work on very large files
43087 by use of a special positioning type.43088 Although *lseek()* may position the file offset beyond the end of the file, this function does not
43089 itself extend the size of the file. While the only function in POSIX.1-2017 that may directly extend

43090 the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally derived from
 43091 the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing calls on *write()*).

43092 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-
 43093 defined and device-dependent (for example, memory may have few invalid values). A negative
 43094 file offset may be valid for some devices in some implementations.

43095 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.
 43096 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return
 43097 to determine whether a return value of (*off_t*)-1 is a negative offset or an indication of an error
 43098 condition. The standard developers did not wish to require this action on the part of a
 43099 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting
 43100 file offset would be negative for a regular file, block special file, or directory.

43101 **FUTURE DIRECTIONS**

43102 None.

43103 **SEE ALSO**

43104 [*open\(\)*](#)

43105 XBD [`<sys/types.h>`](#), [`<unistd.h>`](#)

43106 **CHANGE HISTORY**

43107 First released in Issue 1. Derived from Issue 1 of the SVID.

43108 **Issue 5**

43109 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

43110 Large File Summit extensions are added.

43111 **Issue 6**

43112 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

43113 The following new requirements on POSIX implementations derive from alignment with the
 43114 Single UNIX Specification:

43115 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
 43116 required for conforming implementations of previous POSIX specifications, it was not
 43117 required for UNIX applications.

43118 The [E_OVERFLOW] error condition is added. This change is to support large files.

43119 An additional [ESPIPE] error condition is added for sockets.

43120 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 43121 *lseek()* results are unspecified for typed memory objects.

43122 **Issue 7**

43123 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0366 [421] is applied.

43124 **NAME**

43125 lstat ‡get file status

43126 **SYNOPSIS**

43127 #include <sys/stat.h>

43128 int lstat(const char *restrict *path*, struct stat *restrict *buf*);

43129 **DESCRIPTION**

43130 Refer to *fstatat()*.

43131 **NAME**

43132 malloc ‡'a memory allocator

43133 **SYNOPSIS**

43134 #include <stdlib.h>

43135 void *malloc(size_t size);

43136 **DESCRIPTION**

43137 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43138 conflict between the requirements described here and the ISO C standard is unintentional. This
 43139 volume of POSIX.1-2017 defers to the ISO C standard.

43140 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by
 43141 *size* and whose value is unspecified.

43142 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The
 43143 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 43144 a pointer to any type of object and then used to access such an object in the space allocated (until
 43145 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 43146 disjoint from any other object. The pointer returned points to the start (lowest byte address) of
 43147 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of
 43148 the space requested is 0, the behavior is implementation-defined: either a null pointer shall be
 43149 returned, or the behavior shall be as if the size were some non-zero value, except that the
 43150 behavior is undefined if the returned pointer is used to access an object.

43151 **RETURN VALUE**

43152 Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the
 43153 allocated space. If *size* is 0, either:

43154 CX A null pointer shall be returned and *errno* may be set to an implementation-defined value,
 43155 or

43156 A pointer to the allocated space shall be returned. The application shall ensure that the
 43157 pointer is not used to access an object.

43158 CX Otherwise, it shall return a null pointer and set *errno* to indicate the error.

43159 **ERRORS**

43160 The *malloc()* function shall fail if:

43161 CX [ENOMEM] Insufficient storage space is available.

43162 **EXAMPLES**

43163 None.

43164 **APPLICATION USAGE**

43165 None.

43166 **RATIONALE**

43167 None.

43168 **FUTURE DIRECTIONS**

43169 None.

43170 **SEE ALSO**

43171 *calloc()*, *free()*, *getrlimit()*, *posix_memalign()*, *realloc()*

43172 XBD <stdlib.h>

43173 **CHANGE HISTORY**

43174 First released in Issue 1. Derived from Issue 1 of the SVID.

43175 **Issue 6**

43176 Extensions beyond the ISO C standard are marked.

43177 The following new requirements on POSIX implementations derive from alignment with the
43178 Single UNIX Specification:

43179 In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.

43180 The [ENOMEM] error condition is added.

43181 **Issue 7**

43182 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0203 [526] is applied.

43183 **NAME**43184 `mblen` ‡get number of bytes in a character43185 **SYNOPSIS**43186 `#include <stdlib.h>`43187 `int mblen(const char *s, size_t n);`43188 **DESCRIPTION**

43189 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43190 conflict between the requirements described here and the ISO C standard is unintentional. This
 43191 volume of POSIX.1-2017 defers to the ISO C standard.

43192 If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character
 43193 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

43194 `mbtowc((wchar_t *)0, s, n);`

43195 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 43196 *mblen()*.

43197 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 43198 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 43199 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 43200 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 43201 null pointer shall cause this function to return a non-zero value if encodings have state
 43202 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 43203 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 43204 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 43205 unspecified.

43206 CX The *mblen()* function need not be thread-safe.43207 **RETURN VALUE**

43208 If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,
 43209 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall
 43210 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
 43211 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a
 43212 CX valid character) and may set *errno* to indicate the error. In no case shall the value returned be
 43213 greater than *n* or the value of the {MB_CUR_MAX} macro.

43214 **ERRORS**43215 The *mblen()* function may fail if:

43216 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
 43217 error cannot occur since all byte values are valid characters.

43218 **EXAMPLES**

43219 None.

43220 **APPLICATION USAGE**

43221 None.

43222 **RATIONALE**

43223 None.

43224 **FUTURE DIRECTIONS**

43225 None.

43226 **SEE ALSO**43227 *mbtowc()*, *mbstowcs()*, *wctomb()*, *wcstombs()*43228 XBD **<stdlib.h>**43229 **CHANGE HISTORY**

43230 First released in Issue 4. Aligned with the ISO C standard.

43231 **Issue 7**

43232 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0367 [109] is applied.

43233 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

43234 **NAME**43235 `mbrlen` — get number of bytes in a character (restartable)43236 **SYNOPSIS**43237 `#include <wchar.h>`43238 `size_t mbrlen(const char *restrict s, size_t n,`
43239 `mbstate_t *restrict ps);`43240 **DESCRIPTION**43241 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43242 conflict between the requirements described here and the ISO C standard is unintentional. This
43243 volume of POSIX.1-2017 defers to the ISO C standard.43244 If *s* is not a null pointer, `mbrlen()` shall determine the number of bytes constituting the character
43245 pointed to by *s*. It shall be equivalent to:43246 `mbstate_t internal;`
43247 `mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);`43248 If *ps* is a null pointer, the `mbrlen()` function shall use its own internal `mbstate_t` object, which is
43249 initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object
43250 pointed to by *ps* shall be used to completely describe the current conversion state of the
43251 associated character sequence. The implementation shall behave as if no function defined in this
43252 volume of POSIX.1-2017 calls `mbrlen()`.43253 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.43254 CX The `mbrlen()` function need not be thread-safe if called with a NULL *ps* argument.43255 The `mbrlen()` function shall not change the setting of `errno` if successful.43256 **RETURN VALUE**43257 The `mbrlen()` function shall return the first of the following that applies:43258 0 If the next *n* or fewer bytes complete the character that corresponds to the null
43259 wide character.43260 *positive* If the next *n* or fewer bytes complete a valid character; the value returned shall
43261 be the number of bytes that complete the character.43262 `(size_t)-2` If the next *n* bytes contribute to an incomplete but potentially valid character,
43263 and all *n* bytes have been processed. When *n* has at least the value of the
43264 `{MB_CUR_MAX}` macro, this case can only occur if *s* points at a sequence of
43265 redundant shift sequences (for implementations with state-dependent
43266 encodings).43267 `(size_t)-1` If an encoding error occurs, in which case the next *n* or fewer bytes do not
43268 contribute to a complete and valid character. In this case, `[EILSEQ]` shall be
43269 stored in `errno` and the conversion state is undefined.43270 **ERRORS**43271 The `mbrlen()` function shall fail if:43272 CX `[EILSEQ]` An invalid character sequence is detected. In the POSIX locale an `[EILSEQ]`
43273 error cannot occur since all byte values are valid characters.

- 43274 The *mbrlen()* function may fail if:
- 43275 [EINVAL] *ps* points to an object that contains an invalid conversion state.
- 43276 **EXAMPLES**
- 43277 None.
- 43278 **APPLICATION USAGE**
- 43279 None.
- 43280 **RATIONALE**
- 43281 None.
- 43282 **FUTURE DIRECTIONS**
- 43283 None.
- 43284 **SEE ALSO**
- 43285 *mbstowc()*, *mbtowc()*
- 43286 XBD <[wchar.h](#)>
- 43287 **CHANGE HISTORY**
- 43288 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
- 43289 (E).
- 43290 **Issue 6**
- 43291 The *mbrlen()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.
- 43292 **Issue 7**
- 43293 Austin Group Interpretation 1003.1-2001 #170 is applied.
- 43294 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0368 [109,105] is applied.
- 43295 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

43296 **NAME**43297 `mbrtowc` — convert a character to a wide-character code (restartable)43298 **SYNOPSIS**43299 `#include <wchar.h>`43300 `size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,`
43301 `size_t n, mbstate_t *restrict ps);`43302 **DESCRIPTION**43303 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43304 conflict between the requirements described here and the ISO C standard is unintentional. This
43305 volume of POSIX.1-2017 defers to the ISO C standard.43306 If *s* is a null pointer, the `mbrtowc()` function shall be equivalent to the call:43307 `mbrtowc(NULL, "", 1, ps)`43308 In this case, the values of the arguments *pwc* and *n* are ignored.43309 If *s* is not a null pointer, the `mbrtowc()` function shall inspect at most *n* bytes beginning at the
43310 byte pointed to by *s* to determine the number of bytes needed to complete the next character
43311 (including any shift sequences). If the function determines that the next character is completed,
43312 it shall determine the value of the corresponding wide character and then, if *pwc* is not a null
43313 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide
43314 character is the null wide character, the resulting state described shall be the initial conversion
43315 state.43316 If *ps* is a null pointer, the `mbrtowc()` function shall use its own internal `mbstate_t` object, which
43317 shall be initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t`
43318 object pointed to by *ps* shall be used to completely describe the current conversion state of the
43319 associated character sequence. The implementation shall behave as if no function defined in this
43320 volume of POSIX.1-2017 calls `mbrtowc()`.43321 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.43322 CX The `mbrtowc()` function need not be thread-safe if called with a NULL *ps* argument.43323 The `mbrtowc()` function shall not change the setting of `errno` if successful.43324 **RETURN VALUE**43325 The `mbrtowc()` function shall return the first of the following that applies:43326 0 If the next *n* or fewer bytes complete the character that corresponds to the null
43327 wide character (which is the value stored).43328 between 1 and *n* inclusive43329 If the next *n* or fewer bytes complete a valid character (which is the value
43330 stored); the value returned shall be the number of bytes that complete the
43331 character.43332 (`size_t`)−2 If the next *n* bytes contribute to an incomplete but potentially valid character,
43333 and all *n* bytes have been processed (no value is stored). When *n* has at least
43334 the value of the `{MB_CUR_MAX}` macro, this case can only occur if *s* points at
43335 a sequence of redundant shift sequences (for implementations with state-
43336 dependent encodings).43337 (`size_t`)−1 If an encoding error occurs, in which case the next *n* or fewer bytes do not
43338 contribute to a complete and valid character (no value is stored). In this case,
43339 `[EILSEQ]` shall be stored in `errno` and the conversion state is undefined.

43340 **ERRORS**43341 The *mbrtowc()* function shall fail if:

43342 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
 43343 error cannot occur since all byte values are valid characters.

43344 The *mbrtowc()* function may fail if:

43345 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

43346 **EXAMPLES**

43347 None.

43348 **APPLICATION USAGE**

43349 None.

43350 **RATIONALE**

43351 None.

43352 **FUTURE DIRECTIONS**

43353 None.

43354 **SEE ALSO**43355 [*mbsinit\(\)*](#), [*mbsrtowcs\(\)*](#)43356 XBD <[wchar.h](#)>43357 **CHANGE HISTORY**

43358 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 43359 (E).

43360 **Issue 6**43361 The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43362 The following new requirements on POSIX implementations derive from alignment with the
 43363 Single UNIX Specification:

43364 The [EINVAL] error condition is added.

43365 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

43366 **Issue 7**

43367 Austin Group Interpretation 1003.1-2001 #170 is applied.

43368 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0369 [109,105] is applied.

43369 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0204 [663,674] is applied.

43370 **NAME**43371 `mbsinit` †determine conversion object status43372 **SYNOPSIS**43373 `#include <wchar.h>`43374 `int mbsinit(const mbstate_t *ps);`43375 **DESCRIPTION**

43376 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43377 conflict between the requirements described here and the ISO C standard is unintentional. This
 43378 volume of POSIX.1-2017 defers to the ISO C standard.

43379 If *ps* is not a null pointer, the `mbsinit()` function shall determine whether the object pointed to by
 43380 *ps* describes an initial conversion state.

43381 **RETURN VALUE**

43382 The `mbsinit()` function shall return non-zero if *ps* is a null pointer, or if the pointed-to object
 43383 describes an initial conversion state; otherwise, it shall return zero.

43384 If an `mbstate_t` object is altered by any of the functions described as “restartable”, and is then
 43385 used with a different character sequence, or in the other conversion direction, or with a different
 43386 `LC_CTYPE` category setting than on earlier function calls, the behavior is undefined.

43387 **ERRORS**

43388 No errors are defined.

43389 **EXAMPLES**

43390 None.

43391 **APPLICATION USAGE**

43392 The `mbstate_t` object is used to describe the current conversion state from a particular character
 43393 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the
 43394 `LC_CTYPE` category of the current locale.

43395 The initial conversion state corresponds, for a conversion in either direction, to the beginning of
 43396 a new character sequence in the initial shift state. A zero valued `mbstate_t` object is at least one
 43397 way to describe an initial conversion state. A zero valued `mbstate_t` object can be used to initiate
 43398 conversion involving any character sequence, in any `LC_CTYPE` category setting.

43399 **RATIONALE**

43400 None.

43401 **FUTURE DIRECTIONS**

43402 None.

43403 **SEE ALSO**43404 [mbrlen\(\)](#), [mbrtowc\(\)](#), [mbsrtowcs\(\)](#), [wcrctomb\(\)](#), [wcsrtombs\(\)](#)43405 XBD [<wchar.h>](#)43406 **CHANGE HISTORY**

43407 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 43408 (E).

43409 **NAME**43410 `mbsnrto wcs, mbsrtowcs` — convert a character string to a wide-character string (restartable)43411 **SYNOPSIS**43412 `#include <wchar.h>`

```
43413 CX   size_t mbsnrto wcs(wchar_t *restrict dst, const char **restrict src,
43414         size_t nmc, size_t len, mbstate_t *restrict ps);
43415     size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
43416         size_t len, mbstate_t *restrict ps);
```

43417 **DESCRIPTION**

43418 CX For `mbsrtowcs()`: The functionality described on this reference page is aligned with the ISO C
 43419 standard. Any conflict between the requirements described here and the ISO C standard is
 43420 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

43421 The `mbsrtowcs()` function shall convert a sequence of characters, beginning in the conversion
 43422 state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a
 43423 sequence of corresponding wide characters. If `dst` is not a null pointer, the converted characters
 43424 shall be stored into the array pointed to by `dst`. Conversion continues up to and including a
 43425 terminating null character, which shall also be stored. Conversion shall stop early in either of the
 43426 following cases:

43427 A sequence of bytes is encountered that does not form a valid character.

43428 `len` codes have been stored into the array pointed to by `dst` (and `dst` is not a null pointer).

43429 Each conversion shall take place as if by a call to the `mbrtowc()` function.

43430 If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null
 43431 pointer (if conversion stopped due to reaching a terminating null character) or the address just
 43432 past the last character converted (if any). If conversion stopped due to reaching a terminating
 43433 null character, and if `dst` is not a null pointer, the resulting state described shall be the initial
 43434 conversion state.

43435 If `ps` is a null pointer, the `mbsrtowcs()` function shall use its own internal `mbstate_t` object, which
 43436 is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object
 43437 pointed to by `ps` shall be used to completely describe the current conversion state of the
 43438 associated character sequence.

43439 CX The `mbsnrto wcs()` function shall be equivalent to the `mbsrtowcs()` function, except that the
 43440 conversion of characters indirectly pointed to by `src` is limited to at most `nmc` bytes (the size of
 43441 the input buffer), and under conditions where `mbsrtowcs()` would assign the address just past
 43442 the last character converted (if any) to the pointer object pointed to by `src`, `mbsnrto wcs()` shall
 43443 instead assign the address just past the last byte processed (if any) to that pointer object. If the
 43444 input buffer ends with an incomplete character, it is unspecified whether conversion stops at the
 43445 end of the previous character (if any), or at the end of the input buffer. In the latter case, a
 43446 subsequent call to `mbsnrto wcs()` with an input buffer that starts with the remainder of the
 43447 incomplete character shall correctly complete the conversion of that character.

43448 The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

43449 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 43450 these functions.

43451 CX The `mbsnrto wcs()` and `mbsrtowcs()` functions need not be thread-safe if called with a NULL `ps`
 43452 argument.

43453 The `mbsrtowcs()` function shall not change the setting of `errno` if successful.

43454 **RETURN VALUE**

43455 If the input conversion encounters a sequence of bytes that do not form a valid character, an
 43456 encoding error occurs. In this case, these functions shall store the value of the macro [EILSEQ] in
 43457 *errno* and shall return (**size_t**)-1; the conversion state is undefined. Otherwise, these functions
 43458 shall return the number of characters successfully converted, not including the terminating null
 43459 (if any).

43460 **ERRORS**

43461 These functions shall fail if:

43462 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
 43463 error cannot occur since all byte values are valid characters.

43464 These functions may fail if:

43465 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

43466 **EXAMPLES**

43467 None.

43468 **APPLICATION USAGE**

43469 None.

43470 **RATIONALE**

43471 None.

43472 **FUTURE DIRECTIONS**

43473 A future version may require that when the input buffer ends with an incomplete character,
 43474 conversion stops at the end of the input buffer.

43475 **SEE ALSO**

43476 *iconv()*, *mbrtowc()*, *mbsinit()*

43477 XBD <[wchar.h](#)>

43478 **CHANGE HISTORY**

43479 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 43480 (E).

43481 **Issue 6**

43482 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43483 The [EINVAL] error condition is marked CX.

43484 **Issue 7**

43485 Austin Group Interpretation 1003.1-2001 #170 is applied.

43486 The *mbsnrto wcs()* function is added from The Open Group Technical Standard, 2006, Extended
 43487 API Set Part 1.

43488 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0370 [109,105] is applied.

43489 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0205 [601], XSH/TC2-2008/0206 [663],
 43490 and XSH/TC2-2008/0207 [601] are applied.

43491 **NAME**

43492 mbstowcs †convert a character string to a wide-character string

43493 **SYNOPSIS**

43494 #include <stdlib.h>

43495 size_t mbstowcs(wchar_t *restrict *pwcs*, const char *restrict *s*,
43496 size_t *n*);43497 **DESCRIPTION**43498 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43499 conflict between the requirements described here and the ISO C standard is unintentional. This
43500 volume of POSIX.1-2017 defers to the ISO C standard.43501 The *mbstowcs()* function shall convert a sequence of characters that begins in the initial shift
43502 state from the array pointed to by *s* into a sequence of corresponding wide-character codes and
43503 shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No
43504 characters that follow a null byte (which is converted into a wide-character code with value 0)
43505 shall be examined or converted. Each character shall be converted as if by a call to *mbtowc()*,
43506 except that the shift state of *mbtowc()* is not affected.43507 No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes
43508 place between objects that overlap, the behavior is undefined.43509 XSI The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.
43510 If *pwcs* is a null pointer, *mbstowcs()* shall return the length required to convert the entire array
43511 regardless of the value of *n*, but no values are stored.43512 **RETURN VALUE**43513 CX If an invalid character is encountered, *mbstowcs()* shall return **(size_t)-1** and shall set *errno* to
43514 indicate the error.43515 XSI Otherwise, *mbstowcs()* shall return the number of the array elements modified (or required if
43516 *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if
43517 the value returned is *n*.43518 **ERRORS**43519 The *mbstowcs()* function shall fail if:43520 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
43521 error cannot occur since all byte values are valid characters.43522 **EXAMPLES**

43523 None.

43524 **APPLICATION USAGE**

43525 None.

43526 **RATIONALE**

43527 None.

43528 **FUTURE DIRECTIONS**

43529 None.

43530 **SEE ALSO**43531 *mblen()*, *mbtowc()*, *wctomb()*, *wcstombs()*

43532 XBD <stdlib.h>

43533 **CHANGE HISTORY**

43534 First released in Issue 4. Aligned with the ISO C standard.

43535 **Issue 6**

43536 The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43537 Extensions beyond the ISO C standard are marked.

43538 **Issue 7**

43539 Austin Group Interpretation 1003.1-2001 #170 is applied.

43540 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0371 [195] is applied.

43541 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0208 [663,674] is applied.

43542 **NAME**

43543 mbtowc ‡convert a character to a wide-character code

43544 **SYNOPSIS**

43545 #include <stdlib.h>

43546 int mbtowc(wchar_t *restrict *pwc*, const char *restrict *s*, size_t *n*);43547 **DESCRIPTION**

43548 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43549 conflict between the requirements described here and the ISO C standard is unintentional. This
 43550 volume of POSIX.1-2017 defers to the ISO C standard.

43551 If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the
 43552 character pointed to by *s*. It shall then determine the wide-character code for the value of type
 43553 **wchar_t** that corresponds to that character. (The value of the wide-character code corresponding
 43554 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store
 43555 the wide-character code in the object pointed to by *pwc*.

43556 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 43557 state-dependent encoding, this function is placed into its initial state by a call for which its
 43558 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 43559 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 43560 null pointer shall cause this function to return a non-zero value if encodings have state
 43561 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 43562 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 43563 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 43564 unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

43565 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 43566 *mbtowc()*.

43567 CX The *mbtowc()* function need not be thread-safe.

43568 **RETURN VALUE**

43569 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,
 43570 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*
 43571 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
 43572 CX converted character (if the next *n* or fewer bytes form a valid character), or return -1 and shall
 43573 set *errno* to indicate the error (if they do not form a valid character).

43574 In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

43575 **ERRORS**

43576 The *mbtowc()* function shall fail if:

43577 CX [EILSEQ] An invalid character sequence is detected. In the POSIX locale an [EILSEQ]
 43578 error cannot occur since all byte values are valid characters.

43579 **EXAMPLES**

43580 None.

43581 **APPLICATION USAGE**

43582 None.

43583 **RATIONALE**

43584 None.

43585 **FUTURE DIRECTIONS**

43586 None.

43587 **SEE ALSO**43588 [mblen\(\)](#), [mbstowcs\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)43589 XBD [<stdlib.h>](#)43590 **CHANGE HISTORY**

43591 First released in Issue 4. Aligned with the ISO C standard.

43592 **Issue 6**43593 The `mbtowc()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43594 Extensions beyond the ISO C standard are marked.

43595 **Issue 7**

43596 Austin Group Interpretation 1003.1-2001 #170 is applied.

43597 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0372 [109] and XSH/TC1-2008/0373 [195] are applied.

43599 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0209 [663,674] is applied.

43600 **NAME**

43601 memccpy ‡'copy bytes in memory

43602 **SYNOPSIS**

```
43603 XSI #include <string.h>
43604 void *memccpy(void *restrict s1, const void *restrict s2,
43605 int c, size_t n);
```

43606 **DESCRIPTION**

43607 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first
43608 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,
43609 whichever comes first. If copying takes place between objects that overlap, the behavior is
43610 undefined.

43611 **RETURN VALUE**

43612 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null
43613 pointer if *c* was not found in the first *n* bytes of *s2*.

43614 **ERRORS**

43615 No errors are defined.

43616 **EXAMPLES**

43617 None.

43618 **APPLICATION USAGE**43619 The *memccpy()* function does not check for the overflow of the receiving memory area.43620 **RATIONALE**

43621 None.

43622 **FUTURE DIRECTIONS**

43623 None.

43624 **SEE ALSO**43625 XBD [<string.h>](#)43626 **CHANGE HISTORY**

43627 First released in Issue 1. Derived from Issue 1 of the SVID.

43628 **Issue 6**

43629 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the
43630 ISO/IEC 9899:1999 standard.

43631 **NAME**

43632 memchr ¶find byte in memory

43633 **SYNOPSIS**

43634 #include <string.h>

43635 void *memchr(const void *s, int c, size_t n);

43636 **DESCRIPTION**

43637 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43638 conflict between the requirements described here and the ISO C standard is unintentional. This
43639 volume of POSIX.1-2017 defers to the ISO C standard.

43640 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in
43641 the initial *n* bytes (each interpreted as **unsigned char**) pointed to by *s*.

43642 Implementations shall behave as if they read the memory byte by byte from the beginning of the
43643 bytes pointed to by *s* and stop at the first occurrence of *c* (if it is found in the initial *n* bytes).

43644 **RETURN VALUE**

43645 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte is
43646 not found.

43647 **ERRORS**

43648 No errors are defined.

43649 **EXAMPLES**

43650 None.

43651 **APPLICATION USAGE**

43652 None.

43653 **RATIONALE**

43654 None.

43655 **FUTURE DIRECTIONS**

43656 None.

43657 **SEE ALSO**43658 XBD [<string.h>](#)43659 **CHANGE HISTORY**

43660 First released in Issue 1. Derived from Issue 1 of the SVID.

43661 **Issue 7**

43662 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0374 [110] is applied.

43663 **NAME**

43664 memcmp Compare bytes in memory

43665 **SYNOPSIS**

43666 #include <string.h>

43667 int memcmp(const void *s1, const void *s2, size_t n);

43668 **DESCRIPTION**43669 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
43670 conflict between the requirements described here and the ISO C standard is unintentional. This
43671 volume of POSIX.1-2017 defers to the ISO C standard.43672 The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the
43673 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.43674 The sign of a non-zero return value shall be determined by the sign of the difference between the
43675 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects
43676 being compared.43677 **RETURN VALUE**43678 The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object
43679 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.43680 **ERRORS**

43681 No errors are defined.

43682 **EXAMPLES**

43683 None.

43684 **APPLICATION USAGE**

43685 None.

43686 **RATIONALE**

43687 None.

43688 **FUTURE DIRECTIONS**

43689 None.

43690 **SEE ALSO**43691 XBD [<string.h>](#)43692 **CHANGE HISTORY**

43693 First released in Issue 1. Derived from Issue 1 of the SVID.

43694 **NAME**

43695 memcpy ‡copy bytes in memory

43696 **SYNOPSIS**

43697 #include <string.h>

43698 void *memcpy(void *restrict *s1*, const void *restrict *s2*, size_t *n*);43699 **DESCRIPTION**

43700 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43701 conflict between the requirements described here and the ISO C standard is unintentional. This
43702 volume of POSIX.1-2017 defers to the ISO C standard.

43703 The *memcpy()* function shall copy *n* bytes from the object pointed to by *s2* into the object pointed
43704 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

43705 **RETURN VALUE**43706 The *memcpy()* function shall return *s1*; no return value is reserved to indicate an error.43707 **ERRORS**

43708 No errors are defined.

43709 **EXAMPLES**

43710 None.

43711 **APPLICATION USAGE**43712 The *memcpy()* function does not check for the overflow of the receiving memory area.43713 **RATIONALE**

43714 None.

43715 **FUTURE DIRECTIONS**

43716 None.

43717 **SEE ALSO**43718 XBD <[string.h](#)>43719 **CHANGE HISTORY**

43720 First released in Issue 1. Derived from Issue 1 of the SVID.

43721 **Issue 6**43722 The *memcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

43723 **NAME**

43724 memmove — copy bytes in memory with overlapping areas

43725 **SYNOPSIS**

43726 #include <string.h>

43727 void *memmove(void *s1, const void *s2, size_t n);

43728 **DESCRIPTION**

43729 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43730 conflict between the requirements described here and the ISO C standard is unintentional. This
43731 volume of POSIX.1-2017 defers to the ISO C standard.

43732 The *memmove*() function shall copy *n* bytes from the object pointed to by *s2* into the object
43733 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first
43734 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and
43735 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

43736 **RETURN VALUE**43737 The *memmove*() function shall return *s1*; no return value is reserved to indicate an error.43738 **ERRORS**

43739 No errors are defined.

43740 **EXAMPLES**

43741 None.

43742 **APPLICATION USAGE**

43743 None.

43744 **RATIONALE**

43745 None.

43746 **FUTURE DIRECTIONS**

43747 None.

43748 **SEE ALSO**43749 XBD [<string.h>](#)43750 **CHANGE HISTORY**

43751 First released in Issue 4. Derived from the ANSI C standard.

43752 **NAME**

43753 memset †'set bytes in memory

43754 **SYNOPSIS**

43755 #include <string.h>

43756 void *memset(void *s, int c, size_t n);

43757 **DESCRIPTION**

43758 CX The functionality described on this reference page is aligned with the ISO C standard. Any
43759 conflict between the requirements described here and the ISO C standard is unintentional. This
43760 volume of POSIX.1-2017 defers to the ISO C standard.

43761 The *memset()* function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes
43762 of the object pointed to by *s*.

43763 **RETURN VALUE**43764 The *memset()* function shall return *s*; no return value is reserved to indicate an error.43765 **ERRORS**

43766 No errors are defined.

43767 **EXAMPLES**

43768 None.

43769 **APPLICATION USAGE**

43770 None.

43771 **RATIONALE**

43772 None.

43773 **FUTURE DIRECTIONS**

43774 None.

43775 **SEE ALSO**43776 XBD <[string.h](#)>43777 **CHANGE HISTORY**

43778 First released in Issue 1. Derived from Issue 1 of the SVID.

43779 **NAME**

43780 mkdir, mkdirat — make a directory

43781 **SYNOPSIS**

43782 #include <sys/stat.h>

43783 int mkdir(const char *path, mode_t mode);

43784 OH #include <fcntl.h>

43785 int mkdirat(int fd, const char *path, mode_t mode);

43786 **DESCRIPTION**

43787 The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the
 43788 new directory shall be initialized from *mode*. These file permission bits of the *mode* argument
 43789 shall be modified by the process' file creation mask.

43790 When bits in *mode* other than the file permission bits are set, the meaning of these additional bits
 43791 is implementation-defined.

43792 The directory's user ID shall be set to the process' effective user ID. The directory's group ID
 43793 shall be set to the group ID of the parent directory or to the effective group ID of the process.
 43794 Implementations shall provide a way to initialize the directory's group ID to the group ID of the
 43795 parent directory. Implementations may, but need not, provide an implementation-defined way
 43796 to initialize the directory's group ID to the effective group ID of the calling process.

43797 The newly created directory shall be an empty directory.

43798 If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

43799 Upon successful completion, *mkdir()* shall mark for update the last data access, last data
 43800 modification, and last file status change timestamps of the directory. Also, the last data
 43801 modification and last file status change timestamps of the directory that contains the new entry
 43802 shall be marked for update.

43803 The *mkdirat()* function shall be equivalent to the *mkdir()* function except in the case where *path*
 43804 specifies a relative path. In this case the newly created directory is created relative to the
 43805 directory associated with the file descriptor *fd* instead of the current working directory. If
 43806 the access mode of the open file description associated with the file descriptor is not O_SEARCH,
 43807 the function shall check whether directory searches are permitted using the current permissions
 43808 of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function
 43809 shall not perform the check.

43810 If *mkdirat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 43811 directory shall be used and the behavior shall be identical to a call to *mkdir()*.

43812 **RETURN VALUE**

43813 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 43814 return -1 and set *errno* to indicate the error. If -1 is returned, no directory shall be created.

43815 **ERRORS**

43816 These functions shall fail if:

43817 [EACCES] Search permission is denied on a component of the path prefix, or write
 43818 permission is denied on the parent directory of the directory to be created.

43819 [EEXIST] The named file exists.

43820 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 43821 argument.

43822	[EMLINK]	The link count of the parent directory would exceed {LINK_MAX}.
43823	[ENAMETOOLONG]	
43824		The length of a component of a pathname is longer than {NAME_MAX}.
43825	[ENOENT]	A component of the path prefix specified by <i>path</i> does not name an existing directory or <i>path</i> is an empty string.
43826		
43827	[ENOSPC]	The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.
43828		
43829	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.
43830		
43831	[EROFS]	The parent directory resides on a read-only file system.
43832		In addition, the <i>mkdirat()</i> function shall fail if:
43833	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
43834		
43835		
43836	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
43837		
43838	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
43839		
43840		These functions may fail if:
43841	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
43842		
43843	[ENAMETOOLONG]	
43844		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
43845		
43846		

43847 EXAMPLES

43848 Creating a Directory

43849 The following example shows how to create a directory named **/home/cnd/mod1**, with
 43850 read/write/search permissions for owner and group, and with read/search permissions for
 43851 others.

```

43852 #include <sys/types.h>
43853 #include <sys/stat.h>
43854 int status;
43855 ...
43856 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);

```

43857 APPLICATION USAGE

43858 None.

43859 RATIONALE

43860 The *mkdir()* function originated in 4.2 BSD and was added to System V in Release 3.0.

43861 4.3 BSD detects [ENAMETOOLONG].

43862 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the

43863 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2
 43864 required that implementations provide a way to have the group ID be set to the group ID of the
 43865 containing directory, but did not prohibit implementations also supporting a way to set the
 43866 group ID to the effective group ID of the creating process. Conforming applications should not
 43867 assume which group ID will be used. If it matters, an application can use *chown()* to set the
 43868 group ID after the directory is created, or determine under what conditions the implementation
 43869 will set the desired group ID.

43870 The purpose of the *mkdirat()* function is to create a directory in directories other than the current
 43871 working directory without exposure to race conditions. Any part of the path of a file could be
 43872 changed in parallel to the call to *mkdir()*, resulting in unspecified behavior. By opening a file
 43873 descriptor for the target directory and using the *mkdirat()* function it can be guaranteed that the
 43874 newly created directory is located relative to the desired directory.

43875 **FUTURE DIRECTIONS**

43876 None.

43877 **SEE ALSO**

43878 *chmod()*, *mkdtemp()*, *mkdirat()*, *umask()*

43879 XBD [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)

43880 **CHANGE HISTORY**

43881 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

43882 **Issue 6**

43883 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

43884 The following new requirements on POSIX implementations derive from alignment with the
 43885 Single UNIX Specification:

43886 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 43887 required for conforming implementations of previous POSIX specifications, it was not
 43888 required for UNIX applications.

43889 The [ELOOP] mandatory error condition is added.

43890 A second [ENAMETOOLONG] is added as an optional error condition.

43891 The following changes were made to align with the IEEE P1003.1a draft standard:

43892 The [ELOOP] optional error condition is added.

43893 **Issue 7**

43894 Austin Group Interpretation 1003.1-2001 #143 is applied.

43895 The *mkdirat()* function is added from The Open Group Technical Standard, 2006, Extended API
 43896 Set Part 2.

43897 Changes are made related to support for finegrained timestamps.

43898 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0375 [461], XSH/TC1-2008/0376 [324],
 43899 XSH/TC1-2008/0377 [277], XSH/TC1-2008/0378 [278], and XSH/TC1-2008/0379 [278] are
 43900 applied.

43901 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0210 [873], XSH/TC2-2008/0211 [591],
 43902 XSH/TC2-2008/0212 [817], XSH/TC2-2008/0213 [817], and XSH/TC2-2008/0214 [591] are
 43903 applied.

43904 **NAME**

43905 mkdtemp, mkstemp — create a unique directory or file

43906 **SYNOPSIS**

```
43907 CX #include <stdlib.h>
43908 char *mkdtemp(char *template);
43909 int mkstemp(char *template);
```

43910 **DESCRIPTION**

43911 The *mkdtemp()* function shall create a directory with a unique name derived from *template*. The
 43912 application shall ensure that the string provided in *template* is a pathname ending with at least
 43913 six trailing 'X' characters. The *mkdtemp()* function shall modify the contents of *template* by
 43914 replacing six or more 'X' characters at the end of the pathname with the same number of
 43915 characters from the portable filename character set. The characters shall be chosen such that the
 43916 resulting pathname does not duplicate the name of an existing file at the time of the call to
 43917 *mkdtemp()*. The *mkdtemp()* function shall use the resulting pathname to create the new directory
 43918 as if by a call to:

```
43919 mkdir(pathname, S_IRWXU)
```

43920 The *mkstemp()* function shall create a regular file with a unique name derived from *template* and
 43921 return a file descriptor for the file open for reading and writing. The application shall ensure that
 43922 the string provided in *template* is a pathname ending with at least six trailing 'X' characters. The
 43923 *mkstemp()* function shall modify the contents of *template* by replacing six or more 'X' characters
 43924 at the end of the pathname with the same number of characters from the portable filename
 43925 character set. The characters shall be chosen such that the resulting pathname does not duplicate
 43926 the name of an existing file at the time of the call to *mkstemp()*. The *mkstemp()* function shall use
 43927 the resulting pathname to create the file, and obtain a file descriptor for it, as if by a call to:

```
43928 open(pathname, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)
```

43929 By behaving as if the O_EXCL flag for *open()* is set, the function prevents any possible race
 43930 condition between testing whether the file exists and opening it for use.

43931 **RETURN VALUE**

43932 Upon successful completion, the *mkdtemp()* function shall return the value of *template*.
 43933 Otherwise, it shall return a null pointer and shall set *errno* to indicate the error.

43934 Upon successful completion, the *mkstemp()* function shall return an open file descriptor.
 43935 Otherwise, it shall return -1 and shall set *errno* to indicate the error.

43936 **ERRORS**

43937 The *mkdtemp()* function shall fail if:

- | | | |
|-------|----------------|--|
| 43938 | [EACCES] | Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created. |
| 43939 | | |
| 43940 | [EINVAL] | The string pointed to by <i>template</i> does not end in "XXXXXX". |
| 43941 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the path of the directory to be created. |
| 43942 | | |
| 43943 | [EMLINK] | The link count of the parent directory would exceed {LINK_MAX}. |
| 43944 | [ENAMETOOLONG] | |
| 43945 | | The length of a component of a pathname is longer than {NAME_MAX}. |

43946	[ENOENT]	A component of the path prefix specified by the <i>template</i> argument does not name an existing directory.
43947		
43948	[ENOSPC]	The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.
43949		
43950	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.
43951		
43952	[EROFS]	The parent directory resides on a read-only file system.
43953		The <i>mkdtemp()</i> function may fail if:
43954	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path of the directory to be created.
43955		
43956	[ENAMETOOLONG]	The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
43957		
43958		
43959		
43960		The error conditions for the <i>mkstemp()</i> function are defined in <i>open()</i> .

43961 EXAMPLES

43962 Generating a Pathname

43963 The following example creates a file with a 10-character name beginning with the characters
 43964 "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file
 43965 descriptor that identifies the file.

```
43966 #include <stdlib.h>
43967 ...
43968 char template[] = "/tmp/fileXXXXXX";
43969 int fd;
43970
43971 fd = mkstemp(template);
```

43971 APPLICATION USAGE

43972 It is possible to run out of letters.

43973 Portable applications should pass exactly six trailing 'X's in the template and no more;
 43974 implementations may treat any additional trailing 'X's as either a fixed or replaceable part of
 43975 the template. To be sure of only passing six, a fixed string of at least one non-'X' character
 43976 should precede the six 'X's.

43977 Since 'X' is in the portable filename character set, some of the replacement characters can be
 43978 'X's, leaving part (or even all) of the template effectively unchanged.

43979 RATIONALE

43980 None.

43981 FUTURE DIRECTIONS

43982 None.

43983 SEE ALSO

43984 *getpid()*, *mkdir()*, *open()*, *tmpfile()*, *tmpnam()*

43985 XBD [<stdlib.h>](#)

43986 **CHANGE HISTORY**

43987 First released in Issue 4, Version 2.

43988 **Issue 5**

43989 Moved from X/OPEN UNIX extension to BASE.

43990 **Issue 7**

43991 Austin Group Interpretation 1003.1-2001 #143 is applied.

43992 SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

43993 The *mkstemp()* function is moved from the XSI option to the Base.43994 The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API
43995 Set Part 1.43996 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0380 [291], XSH/TC1-2008/0381 [324],
43997 and XSH/TC1-2008/0382 [291] are applied.

43998 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0215 [567,669] is applied.

43999 **NAME**

44000 mkfifo, mkfifoat ‡make a FIFO special file

44001 **SYNOPSIS**

44002 #include <sys/stat.h>

44003 int mkfifo(const char *path, mode_t mode);

44004 OH #include <fcntl.h>

44005 int mkfifoat(int fd, const char *path, mode_t mode);

44006 **DESCRIPTION**

44007 The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by
 44008 *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission
 44009 bits of the *mode* argument shall be modified by the process' file creation mask.

44010 When bits in *mode* other than the file permission bits are set, the effect is implementation-
 44011 defined.

44012 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].

44013 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set
 44014 to the group ID of the parent directory or to the effective group ID of the process.
 44015 Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the
 44016 parent directory. Implementations may, but need not, provide an implementation-defined way
 44017 to initialize the FIFO's group ID to the effective group ID of the calling process.

44018 Upon successful completion, *mkfifo()* shall mark for update the last data access, last data
 44019 modification, and last file status change timestamps of the file. Also, the last data modification
 44020 and last file status change timestamps of the directory that contains the new entry shall be
 44021 marked for update.

44022 The *mkfifoat()* function shall be equivalent to the *mkfifo()* function except in the case where *path*
 44023 specifies a relative path. In this case the newly created FIFO is created relative to the directory
 44024 associated with the file descriptor *fd* instead of the current working directory. If the access mode
 44025 of the open file description associated with the file descriptor is not O_SEARCH, the function
 44026 shall check whether directory searches are permitted using the current permissions of the
 44027 directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not
 44028 perform the check.

44029 If *mkfifoat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 44030 directory shall be used and the behavior shall be identical to a call to *mkfifo()*.

44031 **RETURN VALUE**

44032 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 44033 return -1 and set *errno* to indicate the error. If -1 is returned, no FIFO shall be created.

44034 **ERRORS**

44035 These functions shall fail if:

44036 [EACCES] A component of the path prefix denies search permission, or write permission
 44037 is denied on the parent directory of the FIFO to be created.

44038 [EEXIST] The named file already exists.

44039 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 44040 argument.

- 44041 [ENAMETOOLONG]
 44042 The length of a component of a pathname is longer than {NAME_MAX}.
- 44043 [ENOENT] A component of the path prefix of *path* does not name an existing file or *path* is
 44044 an empty string.
- 44045 [ENOENT] or [ENOTDIR]
 44046 The *path* argument contains at least one non-`<slash>` character and ends with
 44047 one or more trailing `<slash>` characters. If *path* without the trailing `<slash>`
 44048 characters would name an existing file, an [ENOENT] error shall not occur.
- 44049 [ENOSPC] The directory that would contain the new file cannot be extended or the file
 44050 system is out of file-allocation resources.
- 44051 [ENOTDIR] A component of the path prefix names an existing file that is neither a
 44052 directory nor a symbolic link to a directory.
- 44053 [EROFS] The named file resides on a read-only file system.
- 44054 The *mkfifoat()* function shall fail if:
- 44055 [EACCES] The access mode of the open file description associated with *fd* is not
 44056 O_SEARCH and the permissions of the directory underlying *fd* do not permit
 44057 directory searches.
- 44058 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 44059 neither AT_FDCWD nor a valid file descriptor open for reading or searching.
- 44060 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
 44061 with a non-directory file.
- 44062 These functions may fail if:
- 44063 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 44064 resolution of the *path* argument.
- 44065 [ENAMETOOLONG]
 44066 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
 44067 symbolic link produced an intermediate result with a length that exceeds
 44068 {PATH_MAX}.

44069 EXAMPLES

44070 Creating a FIFO File

44071 The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with
 44072 read/write permissions for owner, and with read permissions for group and others.

```
44073 #include <sys/types.h>
44074 #include <sys/stat.h>

44075 int status;
44076 ...
44077 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
44078               S_IRGRP | S_IROTH);
```

44079 **APPLICATION USAGE**

44080 None.

44081 **RATIONALE**

44082 The syntax of this function is intended to maintain compatibility with historical
 44083 implementations of *mknod()*. The latter function was included in the 1984 /usr/group standard
 44084 but only for use in creating FIFO special files. The *mknod()* function was originally excluded
 44085 from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and
 44086 *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX
 44087 Specification.

44088 The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the
 44089 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2
 44090 required that implementations provide a way to have the group ID be set to the group ID of the
 44091 containing directory, but did not prohibit implementations also supporting a way to set the
 44092 group ID to the effective group ID of the creating process. Conforming applications should not
 44093 assume which group ID will be used. If it matters, an application can use *chown()* to set the
 44094 group ID after the FIFO is created, or determine under what conditions the implementation will
 44095 set the desired group ID.

44096 The purpose of the *mkfifoat()* function is to create a FIFO special file in directories other than the
 44097 current working directory without exposure to race conditions. Any part of the path of a file
 44098 could be changed in parallel to a call to *mkfifo()*, resulting in unspecified behavior. By opening a
 44099 file descriptor for the target directory and using the *mkfifoat()* function it can be guaranteed that
 44100 the newly created FIFO is located relative to the desired directory.

44101 **FUTURE DIRECTIONS**

44102 None.

44103 **SEE ALSO**44104 *chmod()*, *mknod()*, *umask()*44105 XBD [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)44106 **CHANGE HISTORY**

44107 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

44108 **Issue 6**44109 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

44110 The following new requirements on POSIX implementations derive from alignment with the
 44111 Single UNIX Specification:

44112 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 44113 required for conforming implementations of previous POSIX specifications, it was not
 44114 required for UNIX applications.

44115 The [ELOOP] mandatory error condition is added.

44116 A second [ENAMETOOLONG] is added as an optional error condition.

44117 The following changes were made to align with the IEEE P1003.1a draft standard:

44118 The [ELOOP] optional error condition is added.

44119 **Issue 7**

44120 Austin Group Interpretation 1003.1-2001 #143 is applied.

44121 The *mkfifoat()* function is added from The Open Group Technical Standard, 2006, Extended API
 44122 Set Part 2.

- 44123 Changes are made related to support for finegrained timestamps.
- 44124 Changes are made to allow a directory to be opened for searching.
- 44125 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0383 [461], XSH/TC1-2008/0384
44126 [146,435], XSH/TC1-2008/0385 [324], XSH/TC1-2008/0386 [278], and XSH/TC1-2008/0387
44127 [278] are applied.
- 44128 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0216 [873], XSH/TC2-2008/0217 [591],
44129 XSH/TC2-2008/0218 [817], XSH/TC2-2008/0219 [822], XSH/TC2-2008/0220 [817], and
44130 XSH/TC2-2008/0221 [591] are applied.

44131 **NAME**

44132 mknod, mknodat — make directory, special file, or regular file

44133 **SYNOPSIS**

```
44134 XSI #include <sys/stat.h>
44135 int mknod(const char *path, mode_t mode, dev_t dev);

44136 OH XSI #include <fcntl.h>
44137 XSI int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

44138 **DESCRIPTION**

44139 The *mknod()* function shall create a new file named by the pathname to which the argument *path*
44140 points.

44141 The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the
44142 following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

44149 The only portable use of *mknod()* is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is
44150 not 0, the behavior of *mknod()* is unspecified.

44151 The permissions for the new file are OR'ed into the *mode* argument, and may be selected from
44152 any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

44169 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of
44170 the file shall be initialized to either the effective group ID of the process or the group ID of the
44171 parent directory. Implementations shall provide a way to initialize the file's group ID to the
44172 group ID of the parent directory. Implementations may, but need not, provide an
44173 implementation-defined way to initialize the file's group ID to the effective group ID of the

44174 calling process. The owner, group, and other permission bits of *mode* shall be modified by the file
 44175 mode creation mask of the process. The *mknod()* function shall clear each bit whose
 44176 corresponding bit in the file mode creation mask of the process is set.

44177 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].

44178 Upon successful completion, *mknod()* shall mark for update the last data access, last data
 44179 modification, and last file status change timestamps of the file. Also, the last data modification
 44180 and last file status change timestamps of the directory that contains the new entry shall be
 44181 marked for update.

44182 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-
 44183 special.

44184 The *mknodat()* function shall be equivalent to the *mknod()* function except in the case where *path*
 44185 specifies a relative path. In this case the newly created directory, special file, or regular file is
 44186 located relative to the directory associated with the file descriptor *fd* instead of the current
 44187 working directory. If the access mode of the open file description associated with the file
 44188 descriptor is not O_SEARCH, the function shall check whether directory searches are permitted
 44189 using the current permissions of the directory underlying the file descriptor. If the access mode
 44190 is O_SEARCH, the function shall not perform the check.

44191 If *mknodat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 44192 directory shall be used and the behavior shall be identical to a call to *mknod()*.

44193 RETURN VALUE

44194 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 44195 return -1 and set *errno* to indicate the error. If -1 is returned, the new file shall not be created.

44196 ERRORS

44197 These functions shall fail if:

44198 [EACCES] A component of the path prefix denies search permission, or write permission
 44199 is denied on the parent directory.

44200 [EEXIST] The named file exists.

44201 [EINVAL] An invalid argument exists.

44202 [EIO] An I/O error occurred while accessing the file system.

44203 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 44204 argument.

44205 [ENAMETOOLONG]

44206 The length of a component of a pathname is longer than {NAME_MAX}.

44207 [ENOENT] A component of the path prefix of *path* does not name an existing file or *path* is
 44208 an empty string.

44209 [ENOENT] or [ENOTDIR]

44210 The *path* argument contains at least one non-*<slash>* character and ends with
 44211 one or more trailing *<slash>* characters. If *path* without the trailing *<slash>*
 44212 characters would name an existing file, an [ENOENT] error shall not occur.

44213 [ENOSPC] The directory that would contain the new file cannot be extended or the file
 44214 system is out of file allocation resources.

44215	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.
44216		
44217	[EPERM]	The invoking process does not have appropriate privileges and the file type is not FIFO-special.
44218		
44219	[EROFS]	The directory in which the file is to be created is located on a read-only file system.
44220		
44221		The <i>mknodat()</i> function shall fail if:
44222	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
44223		
44224		
44225	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
44226		
44227	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
44228		
44229		These functions may fail if:
44230	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
44231		
44232	[ENAMETOOLONG]	
44233		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
44234		
44235		

44236 EXAMPLES

44237 Creating a FIFO Special File

44238 The following example shows how to create a FIFO special file named `/home/cnd/mod_done`,
44239 with read/write permissions for owner, and with read permissions for group and others.

```
44240 #include <sys/types.h>
44241 #include <sys/stat.h>
44242 dev_t dev;
44243 int status;
44244 ...
44245 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
44246             S_IRUSR | S_IRGRP | S_IROTH, dev);
```

44247 APPLICATION USAGE

44248 The *mkfifo()* function is preferred over this function for making FIFO special files.

44249 RATIONALE

44250 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
44251 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
44252 that implementations provide a way to have the group ID be set to the group ID of the
44253 containing directory, but did not prohibit implementations also supporting a way to set the
44254 group ID to the effective group ID of the creating process. Conforming applications should not
44255 assume which group ID will be used. If it matters, an application can use *chown()* to set the
44256 group ID after the file is created, or determine under what conditions the implementation will
44257 set the desired group ID.

44258 The purpose of the *mknodat()* function is to create directories, special files, or regular files in
44259 directories other than the current working directory without exposure to race conditions. Any
44260 part of the path of a file could be changed in parallel to a call to *mknod()*, resulting in unspecified
44261 behavior. By opening a file descriptor for the target directory and using the *mknodat()* function it
44262 can be guaranteed that the newly created directory, special file, or regular file is located relative
44263 to the desired directory.

44264 **FUTURE DIRECTIONS**

44265 None.

44266 **SEE ALSO**

44267 *chmod()*, *creat()*, *exec*, *fstatat()*, *mkdir()*, *mkfifo()*, *open()*, *umask()*

44268 XBD <*fcntl.h*>, <*sys/stat.h*>

44269 **CHANGE HISTORY**

44270 First released in Issue 4, Version 2.

44271 **Issue 5**

44272 Moved from X/OPEN UNIX extension to BASE.

44273 **Issue 6**

44274 The normative text is updated to avoid use of the term “must” for application requirements.

44275 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
44276 [ELOOP] error condition is added.

44277 **Issue 7**

44278 Austin Group Interpretation 1003.1-2001 #143 is applied.

44279 The *mknodat()* function is added from The Open Group Technical Standard, 2006, Extended API
44280 Set Part 2.

44281 Changes are made related to support for finegrained timestamps.

44282 Changes are made to allow a directory to be opened for searching.

44283 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0388 [324], XSH/TC1-2008/0389 [461],
44284 XSH/TC1-2008/0390 [146,435], XSH/TC1-2008/0391 [278], and XSH/TC1-2008/0392 [278] are
44285 applied.

44286 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0222 [591], XSH/TC2-2008/0223 [817],
44287 XSH/TC2-2008/0224 [822], XSH/TC2-2008/0225 [817], and XSH/TC2-2008/0226 [591] are
44288 applied.

44289 **NAME**

44290 mkstemp — create a unique file

44291 **SYNOPSIS**

```
44292 CX #include <stdlib.h>  
44293 int mkstemp(char *template);
```

44294 **DESCRIPTION**

44295 Refer to *mkdtemp()*.

44296 **NAME**

44297 mktime — convert broken-down time into time since the Epoch

44298 **SYNOPSIS**

44299 #include <time.h>

44300 time_t mktime(struct tm *timeptr);

44301 **DESCRIPTION**

44302 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 44303 conflict between the requirements described here and the ISO C standard is unintentional. This
 44304 volume of POSIX.1-2017 defers to the ISO C standard.

44305 The *mktime()* function shall convert the broken-down time, expressed as local time, in the
 44306 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that
 44307 of the values returned by *time()*. The original values of the *tm_wday* and *tm_yday* components of
 44308 the structure shall be ignored, and the original values of the other components shall not be
 44309 restricted to the ranges described in <time.h>.

44310 CX A positive or 0 value for *tm_isdst* shall cause *mktime()* to presume initially that Daylight Savings
 44311 Time, respectively, is or is not in effect for the specified time. A negative value for *tm_isdst* shall
 44312 cause *mktime()* to attempt to determine whether Daylight Savings Time is in effect for the
 44313 specified time.

44314 Local timezone information shall be set as though *mktime()* called *tzset()*.

44315 The relationship between the **tm** structure (defined in the <time.h> header) and the time in
 44316 seconds since the Epoch is that the result shall be as specified in the expression given in the
 44317 definition of seconds since the Epoch (see XBD Section 4.16, on page 113) corrected for timezone
 44318 and any seasonal time adjustments, where the names other than *tm_yday* in the structure and in
 44319 the expression correspond, and the *tm_yday* value used in the expression is the day of the year
 44320 from 0 to 365 inclusive, calculated from the other **tm** structure members specified in <time.h>
 44321 (excluding *tm_wday*).

44322 Upon successful completion, the values of the *tm_wday* and *tm_yday* components of the structure
 44323 shall be set appropriately, and the other components shall be set to represent the specified time
 44324 since the Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the
 44325 final value of *tm_mday* shall not be set until *tm_mon* and *tm_year* are determined.

44326 **RETURN VALUE**

44327 The *mktime()* function shall return the specified time since the Epoch encoded as a value of type
 44328 **time_t**. If the time since the Epoch cannot be represented, the function shall return the value
 44329 CX (**time_t**)-1 and set *errno* to indicate the error.

44330 **ERRORS**

44331 The *mktime()* function shall fail if:

44332 CX [EOVERFLOW] The result cannot be represented.

44333 **EXAMPLES**

44334 What day of the week is July 4, 2001?

44335 #include <stdio.h>

44336 #include <time.h>

44337 struct tm time_str;

44338 char daybuf[20];

44339 int main(void)

44340 {

44341 time_str.tm_year = 2001 - 1900;

44342 time_str.tm_mon = 7 - 1;

44343 time_str.tm_mday = 4;

44344 time_str.tm_hour = 0;

44345 time_str.tm_min = 0;

44346 time_str.tm_sec = 1;

44347 time_str.tm_isdst = -1;

44348 if (mktime(&time_str) == -1)

44349 (void)puts("-unknown-");

44350 else {

44351 (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);

44352 (void)puts(daybuf);

44353 }

44354 return 0;

44355 }

44356 **APPLICATION USAGE**

44357 None.

44358 **RATIONALE**

44359 None.

44360 **FUTURE DIRECTIONS**

44361 None.

44362 **SEE ALSO**44363 *asctime()*, *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *strftime()*, *strptime()*, *time()*, *tzset()*,
44364 *utime()*44365 XBD Section 4.16 (on page 113), <[time.h](#)>44366 **CHANGE HISTORY**44367 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C
44368 standard.44369 **Issue 6**

44370 Extensions beyond the ISO C standard are marked.

44371 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/58 is applied, updating the RETURN
44372 VALUE and ERRORS sections to add the optional [EOVERFLOW] error as a CX extension.44373 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/59 is applied, adding the *tzset()* function
44374 to the SEE ALSO section.

44375 **Issue 7**

44376 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0393 [104] is applied.

44377 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0228 [724] is applied.

44378 **NAME**44379 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)44380 **SYNOPSIS**

```
44381 MLR #include <sys/mman.h>
44382 int mlock(const void *addr, size_t len);
44383 int munlock(const void *addr, size_t len);
```

44384 **DESCRIPTION**

44385 The *mlock()* function shall cause those whole pages containing any part of the address space of
 44386 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until
 44387 unlocked or until the process exits or *execs* another process image. The implementation may
 44388 require that *addr* be a multiple of {PAGESIZE}.

44389 The *munlock()* function shall unlock those whole pages containing any part of the address space
 44390 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many
 44391 times *mlock()* has been called by the process for any of the pages in the specified range. The
 44392 implementation may require that *addr* be a multiple of {PAGESIZE}.

44393 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address
 44394 spaces of other processes, any locks established on those pages by another process are
 44395 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a
 44396 call to *munlock()* are also mapped into other portions of the address space of the calling process
 44397 outside the range specified, any locks established on those pages via the other mappings are also
 44398 unaffected by this call.

44399 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-
 44400 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked
 44401 with respect to the address space of the process. Memory residency of unlocked pages is
 44402 unspecified.

44403 Appropriate privileges are required to lock process memory with *mlock()*.

44404 **RETURN VALUE**

44405 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.
 44406 Otherwise, no change is made to any locks in the address space of the process, and the function
 44407 shall return a value of -1 and set *errno* to indicate the error.

44408 **ERRORS**

44409 The *mlock()* and *munlock()* functions shall fail if:

44410 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does
 44411 not correspond to valid mapped pages in the address space of the process.

44412 The *mlock()* function shall fail if:

44413 [EAGAIN] Some or all of the memory identified by the operation could not be locked
 44414 when the call was made.

44415 The *mlock()* and *munlock()* functions may fail if:

44416 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

44417 The *mlock()* function may fail if:

44418 [ENOMEM] Locking the pages mapped by the specified range would exceed an
 44419 implementation-defined limit on the amount of memory that the process may
 44420 lock.

44421 [EPERM] The calling process does not have appropriate privileges to perform the
44422 requested operation.

44423 **EXAMPLES**

44424 None.

44425 **APPLICATION USAGE**

44426 None.

44427 **RATIONALE**

44428 None.

44429 **FUTURE DIRECTIONS**

44430 None.

44431 **SEE ALSO**

44432 *exec, exit(), fork(), mlockall(), munmap()*

44433 XBD <sys/mman.h>

44434 **CHANGE HISTORY**

44435 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

44436 **Issue 6**

44437 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.

44438 The [ENOSYS] error condition has been removed as stubs need not be provided if an
44439 implementation does not support the Range Memory Locking option.

44440 **NAME**44441 mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)44442 **SYNOPSIS**

```
44443 ML #include <sys/mman.h>
44444 int mlockall(int flags);
44445 int munlockall(void);
```

44446 **DESCRIPTION**

44447 The *mlockall()* function shall cause all of the pages mapped by the address space of a process to
 44448 be memory-resident until unlocked or until the process exits or *execs* another process image. The
 44449 *flags* argument determines whether the pages to be locked are those currently mapped by the
 44450 address space of the process, those that are mapped in the future, or both. The *flags* argument is
 44451 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,
 44452 defined in *<sys/mman.h>*:

44453 **MCL_CURRENT** Lock all of the pages currently mapped into the address space of the process.

44454 **MCL_FUTURE** Lock all of the pages that become mapped into the address space of the
 44455 process in the future, when those mappings are established.

44456 If **MCL_FUTURE** is specified, and the automatic locking of future mappings eventually causes
 44457 the amount of locked memory to exceed the amount of available physical memory or any other
 44458 implementation-defined limit, the behavior is implementation-defined. The manner in which the
 44459 implementation informs the application of these situations is also implementation-defined.

44460 The *munlockall()* function shall unlock all currently mapped pages of the address space of the
 44461 process. Any pages that become mapped into the address space of the process after a call to
 44462 *munlockall()* shall not be locked, unless there is an intervening call to *mlockall()* specifying
 44463 **MCL_FUTURE** or a subsequent call to *mlockall()* specifying **MCL_CURRENT**. If pages mapped
 44464 into the address space of the process are also mapped into the address spaces of other processes
 44465 and are locked by those processes, the locks established by the other processes shall be
 44466 unaffected by a call by this process to *munlockall()*.

44467 Upon successful return from the *mlockall()* function that specifies **MCL_CURRENT**, all currently
 44468 mapped pages of the address space of the process shall be memory-resident and locked. Upon
 44469 return from the *munlockall()* function, all currently mapped pages of the address space of the
 44470 process shall be unlocked with respect to the address space of the process. The memory
 44471 residency of unlocked pages is unspecified.

44472 Appropriate privileges are required to lock process memory with *mlockall()*.

44473 **RETURN VALUE**

44474 Upon successful completion, the *mlockall()* function shall return a value of zero. Otherwise, no
 44475 additional memory shall be locked, and the function shall return a value of -1 and set *errno* to
 44476 indicate the error. The effect of failure of *mlockall()* on previously existing locks in the address
 44477 space is unspecified.

44478 If it is supported by the implementation, the *munlockall()* function shall always return a value of
 44479 zero. Otherwise, the function shall return a value of -1 and set *errno* to indicate the error.

44480 **ERRORS**

44481 The *mlockall()* function shall fail if:

44482 **[EAGAIN]** Some or all of the memory identified by the operation could not be locked
 44483 when the call was made.

44484 [EINVAL] The *flags* argument is zero, or includes unimplemented flags.

44485 The *mlockall()* function may fail if:

44486 [ENOMEM] Locking all of the pages currently mapped into the address space of the
44487 process would exceed an implementation-defined limit on the amount of
44488 memory that the process may lock.

44489 [EPERM] The calling process does not have appropriate privileges to perform the
44490 requested operation.

44491 **EXAMPLES**

44492 None.

44493 **APPLICATION USAGE**

44494 None.

44495 **RATIONALE**

44496 None.

44497 **FUTURE DIRECTIONS**

44498 None.

44499 **SEE ALSO**

44500 *exec*, *exit()*, *fork()*, *mlock()*, *munmap()*

44501 XBD <[sys/mman.h](#)>

44502 **CHANGE HISTORY**

44503 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

44504 **Issue 6**

44505 The *mlockall()* and *munlockall()* functions are marked as part of the Process Memory Locking
44506 option.

44507 The [ENOSYS] error condition has been removed as stubs need not be provided if an
44508 implementation does not support the Process Memory Locking option.

44509 **NAME**44510 `mmap` †map pages of memory44511 **SYNOPSIS**44512 `#include <sys/mman.h>`44513 `void *mmap(void *addr, size_t len, int prot, int flags,`
44514 `int fildes, off_t off);`44515 **DESCRIPTION**44516 The `mmap()` function shall establish a mapping between an address space of a process and a
44517 memory object.44518 The `mmap()` function shall be supported for the following memory objects:

44519 Regular files

44520 SHM Shared memory objects

44521 TYM Typed memory objects

44522 Support for any other type of file is unspecified.

44523 The format of the call is as follows:

44524 `pa=mmap(addr, len, prot, flags, fildes, off);`44525 The `mmap()` function shall establish a mapping between the address space of the process at an
44526 address `pa` for `len` bytes to the memory object represented by the file descriptor `fildes` at offset `off`
44527 for `len` bytes. The value of `pa` is an implementation-defined function of the parameter `addr` and
44528 the values of `flags`, further described below. A successful `mmap()` call shall return `pa` as its result.
44529 The address range starting at `pa` and continuing for `len` bytes shall be legitimate for the possible
44530 (not necessarily current) address space of the process. The range of bytes starting at `off` and
44531 continuing for `len` bytes shall be legitimate for the possible (not necessarily current) offsets in the
44532 memory object represented by `fildes`.44533 TYM If `fildes` represents a typed memory object opened with either the
44534 `POSIX_TYPED_MEM_ALLOCATE` flag or the `POSIX_TYPED_MEM_ALLOCATE_CONTIG`
44535 flag, the memory object to be mapped shall be that portion of the typed memory object allocated
44536 by the implementation as specified below. In this case, if `off` is non-zero, the behavior of `mmap()`
44537 is undefined. If `fildes` refers to a valid typed memory object that is not accessible from the calling
44538 process, `mmap()` shall fail.44539 The mapping established by `mmap()` shall replace any previous mappings for those whole pages
44540 containing any part of the address space of the process starting at `pa` and continuing for `len`
44541 bytes.44542 If the size of the mapped file changes after the call to `mmap()` as a result of some other operation
44543 on the mapped file, the effect of references to portions of the mapped region that correspond to
44544 added or removed portions of the file is unspecified.44545 If `len` is zero, `mmap()` shall fail and no mapping shall be established.44546 The parameter `prot` determines whether read, write, execute, or some combination of accesses
44547 are permitted to the data being mapped. The `prot` shall be either `PROT_NONE` or the bitwise-
44548 inclusive OR of one or more of the other flags in the following table, defined in the
44549 `<sys/mman.h>` header.

44550
44551
44552
44553
44554

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

44555
44556

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap()* shall fail.

44557
44558
44559
44560
44561
44562
44563
44564
44565

An implementation may permit accesses other than those specified by *prot*; however, the implementation shall not permit a write to succeed where PROT_WRITE has not been set and shall not permit any access where PROT_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. The file descriptor *fildev* shall have been opened with read permission, regardless of the protection options specified. If PROT_WRITE is specified, the application shall ensure that it has opened the file descriptor *fildev* with write permission unless MAP_PRIVATE is specified in the *flags* parameter as described below.

44566
44567

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

44568
44569
44570
44571

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

44572 XSI
44573

It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be supported on XSI-conformant systems.

44574
44575
44576
44577
44578
44579
44580

MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory object. If MAP_SHARED is specified, write references shall change the underlying object. If MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP_PRIVATE mapping is established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork()*.

44581
44582
44583

The state of synchronization objects such as mutexes, semaphores, barriers, and conditional variables placed in shared memory mapped with MAP_SHARED becomes undefined when the last region in any process containing the synchronization object is unmapped.

44584 TYM
44585
44586
44587
44588
44589
44590
44591
44592
44593
44594
44595
44596

When *fildev* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, *mmap()* shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *fildev* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *fildev* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *fildev* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory object are mapped, exactly as when mapping a file or shared memory object. In this case, if two

44597 processes map an area of typed memory using the same *off* and *len* values and using file
 44598 descriptors that refer to the same memory pool (either from the same port or from a different
 44599 port), both processes shall map the same region of storage.

44600 When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of
 44601 *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap()* may return MAP_FAILED and set *errno* to
 44602 MLIMLR [EINVAL]. If a MAP_FIXED request is successful, then any previous mappings or memory
 44603 locks for those whole pages containing any part of the address range [*pa*,*pa+len*) shall be
 44604 removed, as if by an appropriate call to *munmap()*, before the new mapping is established.

44605 When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined
 44606 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the
 44607 implementation deems suitable for a mapping of *len* bytes to the file. All implementations
 44608 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,
 44609 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a
 44610 process address near which the mapping should be placed. When the implementation selects a
 44611 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

44612 If MAP_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off*
 44613 parameter, modulo the page size as returned by *sysconf()* when passed *_SC_PAGESIZE* or
 44614 *_SC_PAGE_SIZE*. The implementation may require that *off* is a multiple of the page size. If
 44615 MAP_FIXED is specified, the implementation may require that *addr* is a multiple of the page
 44616 size. The system performs mapping operations over whole pages. Thus, while the parameter *len*
 44617 need not meet a size or alignment constraint, the system shall include, in any mapping
 44618 operation, any partial page specified by the address range starting at *pa* and continuing for *len*
 44619 bytes.

44620 The system shall always zero-fill any partial page at the end of an object. Further, the system
 44621 shall never write out any modified portions of the last page of an object which are beyond its
 44622 end. References within the address range starting at *pa* and continuing for *len* bytes to whole
 44623 pages following the end of an object shall result in delivery of a SIGBUS signal.

44624 An implementation may generate SIGBUS signals when a reference would cause an error in the
 44625 mapped object, such as out-of-space condition.

44626 The *mmap()* function shall add an extra reference to the file associated with the file descriptor
 44627 *fd* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be
 44628 removed when there are no more mappings to the file.

44629 The last data access timestamp of the mapped file may be marked for update at any time
 44630 between the *mmap()* call and the corresponding *munmap()* call. The initial read or write
 44631 reference to a mapped region shall cause the file's last data access timestamp to be marked for
 44632 update if it has not already been marked for update.

44633 The last data modification and last file status change timestamps of a file that is mapped with
 44634 MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval
 44635 between a write reference to the mapped region and the next call to *msync()* with MS_ASYNC or
 44636 MS_SYNC for that portion of the file by any process. If there is no such call and if the
 44637 underlying file is modified as a result of a write reference, then these timestamps shall be
 44638 marked for update at some time after the write reference.

44639 There may be implementation-defined limits on the number of memory regions that can be
 44640 mapped (per process or per system).

44641 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a
 44642 process is decreased by the use of *shmat()* is implementation-defined.

44643 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the
 44644 mappings in the address range starting at *addr* and continuing for *len* bytes may have been
 44645 unmapped.

44646 RETURN VALUE

44647 Upon successful completion, the *mmap()* function shall return the address at which the mapping
 44648 was placed (*pa*); otherwise, it shall return a value of MAP_FAILED and set *errno* to indicate the
 44649 error. The symbol MAP_FAILED is defined in the <sys/mman.h> header. No successful return
 44650 from *mmap()* shall return the value MAP_FAILED.

44651 ERRORS

44652 The *mmap()* function shall fail if:

44653 [EACCES] The *fildev* argument is not open for read, regardless of the protection specified,
 44654 or *fildev* is not open for write and PROT_WRITE was specified for a
 44655 MAP_SHARED type mapping.

44656 ML [EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to
 44657 a lack of resources.

44658 [EBADF] The *fildev* argument is not a valid open file descriptor.

44659 [EINVAL] The value of *len* is zero.

44660 [EINVAL] The value of *flags* is invalid (neither MAP_PRIVATE nor MAP_SHARED is
 44661 set).

44662 [EMFILE] The number of mapped regions would exceed an implementation-defined
 44663 limit (per process or per system).

44664 [ENODEV] The *fildev* argument refers to a file whose type is not supported by *mmap()*.

44665 [ENOMEM] MAP_FIXED was specified, and the range [*addr,addr+len*) exceeds that allowed
 44666 for the address space of a process; or, if MAP_FIXED was not specified and
 44667 there is insufficient room in the address space to effect the mapping.

44668 ML [ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*,
 44669 because it would require more space than the system is able to supply.

44670 TYM [ENOMEM] Not enough unallocated memory resources remain in the typed memory
 44671 object designated by *fildev* to allocate *len* bytes.

44672 [ENOTSUP] MAP_FIXED or MAP_PRIVATE was specified in the *flags* argument and the
 44673 implementation does not support this functionality.

44674 The implementation does not support the combination of accesses requested
 44675 in the *prot* argument.

44676 [ENXIO] Addresses in the range [*off,off+len*) are invalid for the object specified by *fildev*.

44677 [ENXIO] MAP_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is
 44678 invalid for the object specified by *fildev*.

44679 TYM [ENXIO] The *fildev* argument refers to a typed memory object that is not accessible from
 44680 the calling process.

44681 [EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset
 44682 maximum established in the open file description associated with *fildev*.

44683 The *mmap()* function may fail if:

44684 [EINVAL] The *addr* argument (if MAP_FIXED was specified) or *off* is not a multiple of the
 44685 page size as returned by *sysconf()*, or is considered invalid by the
 44686 implementation.

44687 EXAMPLES

44688 None.

44689 APPLICATION USAGE

44690 Use of *mmap()* may reduce the amount of memory available to other memory allocation
 44691 functions.

44692 Use of MAP_FIXED may result in unspecified behavior in further use of *malloc()* and *shmat()*.
 44693 The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the
 44694 most effective use of resources. Most implementations require that *off* and *addr* are multiples of
 44695 the page size as returned by *sysconf()*.

44696 The application must ensure correct synchronization when using *mmap()* in conjunction with
 44697 any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

44698 The *mmap()* function allows access to resources via address space manipulations, instead of
 44699 *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the
 44700 address to which the file was mapped. So, using pseudo-code to illustrate the way in which an
 44701 existing program might be changed to use *mmap()*, the following:

```
44702 fildes = open(...)
44703 lseek(fildes, some_offset)
44704 read(fildes, buf, len)
44705 /* Use data in buf. */
```

44706 becomes:

```
44707 fildes = open(...)
44708 address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
44709 /* Use data at address. */
```

44710 RATIONALE

44711 After considering several other alternatives, it was decided to adopt the *mmap()* definition
 44712 found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is
 44713 minimal, in that it describes only what has been built, and what appears to be necessary for a
 44714 general and portable mapping facility.

44715 Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose
 44716 mapping facility. It can be used to map any appropriate object, such as memory, files, devices,
 44717 and so on, into the address space of a process.

44718 When a mapping is established, it is possible that the implementation may need to map more
 44719 than is requested into the address space of the process because of hardware requirements. An
 44720 application, however, cannot count on this behavior. Implementations that do not use a paged
 44721 architecture may simply allocate a common memory region and return the address of it; such
 44722 implementations probably do not allocate any more than is necessary. References past the end of
 44723 the requested area are unspecified.

44724 If an application requests a mapping that overlaps existing mappings in the process, it might be
 44725 desirable that an implementation detect this and inform the application. However, if the
 44726 program specifies a fixed address mapping (which requires some implementation knowledge to
 44727 determine a suitable address, if the function is supported at all), then the program is presumed

44728 to be successfully managing its own address space and should be trusted when it asks to map
44729 over existing data structures. Furthermore, it is also desirable to make as few system calls as
44730 possible, and it might be considered onerous to require an *munmap()* before an *mmap()* to the
44731 same address range. This volume of POSIX.1-2017 specifies that the new mapping replaces any
44732 existing mappings (implying an automatic *munmap()* on the address range), following existing
44733 practice in this regard. The standard developers also considered whether there should be a way
44734 for new mappings to overlay existing mappings, but found no existing practice for this.

44735 It is not expected that all hardware implementations are able to support all combinations of
44736 permissions at all addresses. Implementations are required to disallow write access to mappings
44737 without write permission and to disallow access to mappings without any access permission.
44738 Other than these restrictions, implementations may allow access types other than those
44739 requested by the application. For example, if the application requests only *PROT_WRITE*, the
44740 implementation may also allow read access. A call to *mmap()* fails if the implementation cannot
44741 support allowing all the access requested by the application. For example, some
44742 implementations cannot support a request for both write access and execute access
44743 simultaneously. All implementations must support requests for no access, read access, write
44744 access, and both read and write access. Strictly conforming code must only rely on the required
44745 checks. These restrictions allow for portability across a wide range of hardware.

44746 The *MAP_FIXED* address treatment is likely to fail for non-page-aligned values and for certain
44747 architecture-dependent address ranges. Conforming implementations cannot count on being
44748 able to choose address values for *MAP_FIXED* without utilizing non-portable, implementation-
44749 defined knowledge. Nonetheless, *MAP_FIXED* is provided as a standard interface conforming
44750 to existing practice for utilizing such knowledge when it is available.

44751 Similarly, in order to allow implementations that do not support virtual addresses, support for
44752 directly specifying any mapping addresses via *MAP_FIXED* is not required and thus a
44753 conforming application may not count on it.

44754 The *MAP_PRIVATE* function can be implemented efficiently when memory protection hardware
44755 is available. When such hardware is not available, implementations can implement such
44756 “mappings” by simply making a real copy of the relevant data into process private memory,
44757 though this tends to behave similarly to *read()*.

44758 The function has been defined to allow for many different models of using shared memory.
44759 However, all uses are not equally portable across all machine architectures. In particular, the
44760 *mmap()* function allows the system as well as the application to specify the address at which to
44761 map a specific region of a memory object. The most portable way to use the function is always to
44762 let the system choose the address, specifying *NULL* as the value for the argument *addr* and not
44763 to specify *MAP_FIXED*.

44764 If it is intended that a particular region of a memory object be mapped at the same address in a
44765 group of processes (on machines where this is even possible), then *MAP_FIXED* can be used to
44766 pass in the desired mapping address. The system can still be used to choose the desired address
44767 if the first such mapping is made without specifying *MAP_FIXED*, and then the resulting
44768 mapping address can be passed to subsequent processes for them to pass in via *MAP_FIXED*.
44769 The availability of a specific address range cannot be guaranteed, in general.

44770 The *mmap()* function can be used to map a region of memory that is larger than the current size
44771 of the object. Memory access within the mapping but beyond the current end of the underlying
44772 objects may result in *SIGBUS* signals being sent to the process. The reason for this is that the size
44773 of the object can be manipulated by other processes and can change at any moment. The
44774 implementation should tell the application that a memory reference is outside the object where
44775 this can be detected; otherwise, written data may be lost and read data may not reflect actual

- 44776 data in the object.
- 44777 Note that references beyond the end of the object do not extend the object as the new end cannot
44778 be determined precisely by most virtual memory hardware. Instead, the size can be directly
44779 manipulated by *ftruncate()*.
- 44780 Process memory locking does apply to shared memory regions, and the MCL_FUTURE
44781 argument to *mlockall()* can be relied upon to cause new shared memory regions to be
44782 automatically locked.
- 44783 Existing implementations of *mmap()* return the value `-1` when unsuccessful. Since the casting of
44784 this value to type `void *` cannot be guaranteed by the ISO C standard to be distinct from a
44785 successful value, this volume of POSIX.1-2017 defines the symbol `MAP_FAILED`, which a
44786 conforming implementation does not return as the result of a successful call.
- 44787 **FUTURE DIRECTIONS**
- 44788 None.
- 44789 **SEE ALSO**
- 44790 *exec*, *fcntl()*, *fork()*, *lockf()*, *msync()*, *munmap()*, *mprotect()*, *posix_typed_mem_open()*, *shmat()*,
44791 *sysconf()*
- 44792 XBD [<sys/mman.h>](#)
- 44793 **CHANGE HISTORY**
- 44794 First released in Issue 4, Version 2.
- 44795 **Issue 5**
- 44796 Moved from X/OPEN UNIX extension to BASE.
- 44797 Aligned with *mmap()* in the POSIX Realtime Extension as follows:
- 44798 The DESCRIPTION is extensively reworded.
- 44799 The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 44800 New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 44801 The value returned on failure is the value of the constant `MAP_FAILED`; this was
44802 previously defined as `-1`.
- 44803 Large File Summit extensions are added.
- 44804 **Issue 6**
- 44805 The *mmap()* function is marked as part of the Memory Mapped Files option.
- 44806 The Open Group Corrigendum U028/6 is applied, changing `(void *)-1` to `MAP_FAILED`.
- 44807 The following new requirements on POSIX implementations derive from alignment with the
44808 Single UNIX Specification:
- 44809 The DESCRIPTION is updated to describe the use of `MAP_FIXED`.
- 44810 The DESCRIPTION is updated to describe the addition of an extra reference to the file
44811 associated with the file descriptor passed to *mmap()*.
- 44812 The DESCRIPTION is updated to state that there may be implementation-defined limits on
44813 the number of memory regions that can be mapped.
- 44814 The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*
44815 argument.

- 44816 The [EINVAL] and [EMFILE] error conditions are added.
- 44817 The [EOVERFLOW] error condition is added. This change is to support large files.
- 44818 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:
- 44819 The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and
44820 MAP_FIXED need not be supported.
- 44821 The following changes are made for alignment with IEEE Std 1003.1j-2000:
- 44822 Semantics for typed memory objects are added to the DESCRIPTION.
- 44823 New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 44824 The *posix_typed_mem_open()* function is added to the SEE ALSO section.
- 44825 The normative text is updated to avoid use of the term “must” for application requirements.
- 44826 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code
44827 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).
- 44828 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/60 is applied, updating the
44829 DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.
- 44830 **Issue 7**
- 44831 Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment
44832 requirements and adding a note about the state of synchronization objects becoming undefined
44833 when a shared region is unmapped.
- 44834 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
44835 the Base.
- 44836 Changes are made related to support for finegrained timestamps.
- 44837 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0229 [852] is applied.

44838 **NAME**

44839 modf, modff, modfl — decompose a floating-point number

44840 **SYNOPSIS**

44841 #include <math.h>

44842 double modf(double *x*, double **iptr*);44843 float modff(float *value*, float **iptr*);44844 long double modfl(long double *value*, long double **iptr*);44845 **DESCRIPTION**

44846 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 44847 conflict between the requirements described here and the ISO C standard is unintentional. This
 44848 volume of POSIX.1-2017 defers to the ISO C standard.

44849 These functions shall break the argument *x* into integral and fractional parts, each of which has
 44850 the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a
 44851 **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed
 44852 to by *iptr*.

44853 **RETURN VALUE**44854 Upon successful completion, these functions shall return the signed fractional part of *x*.44855 MX If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.44856 If *x* is $\pm\text{Inf}$, ± 0 shall be returned, and **iptr* shall be set to $\pm\text{Inf}$.44857 **ERRORS**

44858 No errors are defined.

44859 **EXAMPLES**

44860 None.

44861 **APPLICATION USAGE**44862 The *modf()* function computes the function result and **iptr* such that:44863 `a = modf(x, iptr) ;`44864 `x == a+*iptr ;`

44865 allowing for the usual floating-point inaccuracies.

44866 **RATIONALE**

44867 None.

44868 **FUTURE DIRECTIONS**

44869 None.

44870 **SEE ALSO**44871 *frexp()*, *isnan()*, *ldexp()*

44872 XBD <math.h>

44873 **CHANGE HISTORY**

44874 First released in Issue 1. Derived from Issue 1 of the SVID.

44875 **Issue 5**

44876 The DESCRIPTION is updated to indicate how an application should check for an error. This
 44877 text was previously published in the APPLICATION USAGE section.

44878 **Issue 6**

44879 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999
44880 standard.

44881 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
44882 revised to align with the ISO/IEC 9899:1999 standard.

44883 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
44884 marked.

44885 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example
44886 in the APPLICATION USAGE section.

44887 **NAME**

44888 mprotect — set protection of memory mapping

44889 **SYNOPSIS**

44890 #include <sys/mman.h>

44891 int mprotect(void *addr, size_t len, int prot);

44892 **DESCRIPTION**

44893 The *mprotect()* function shall change the access protections to be that specified by *prot* for those
 44894 whole pages containing any part of the address space of the process starting at address *addr* and
 44895 continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some
 44896 combination of accesses are permitted to the data being mapped. The *prot* argument should be
 44897 either PROT_NONE or the bitwise-inclusive OR of one or more of PROT_READ, PROT_WRITE,
 44898 and PROT_EXEC.

44899 If an implementation cannot support the combination of access types specified by *prot*, the call to
 44900 *mprotect()* shall fail.

44901 An implementation may permit accesses other than those specified by *prot*; however, no
 44902 implementation shall permit a write to succeed where PROT_WRITE has not been set or shall
 44903 permit any access where PROT_NONE alone has been set. Implementations shall support at
 44904 least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-
 44905 inclusive OR of PROT_READ and PROT_WRITE. If PROT_WRITE is specified, the application
 44906 shall ensure that it has opened the mapped objects in the specified address range with write
 44907 permission, unless MAP_PRIVATE was specified in the original mapping, regardless of whether
 44908 the file descriptors used to map the objects have since been closed.

44909 The implementation may require that *addr* be a multiple of the page size as returned by
 44910 *sysconf()*.

44911 The behavior of this function is unspecified if the mapping was not established by a call to
 44912 *mmap()*.

44913 When *mprotect()* fails for reasons other than [EINVAL], the protections on some of the pages in
 44914 the range [*addr,addr+len*) may have been changed.

44915 **RETURN VALUE**

44916 Upon successful completion, *mprotect()* shall return 0; otherwise, it shall return -1 and set *errno*
 44917 to indicate the error.

44918 **ERRORS**

44919 The *mprotect()* function shall fail if:

44920 [EACCES] The *prot* argument specifies a protection that violates the access permission the
 44921 process has to the underlying memory object.

44922 [EAGAIN] The *prot* argument specifies PROT_WRITE over a MAP_PRIVATE mapping
 44923 and there are insufficient memory resources to reserve for locking the private
 44924 page.

44925 [ENOMEM] Addresses in the range [*addr,addr+len*) are invalid for the address space of a
 44926 process, or specify one or more pages which are not mapped.

44927 [ENOMEM] The *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and
 44928 it would require more space than the system is able to supply for locking the
 44929 private pages, if required.

44930 [ENOTSUP] The implementation does not support the combination of accesses requested
44931 in the *prot* argument.

44932 The *mprotect()* function may fail if:

44933 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

44934 **EXAMPLES**

44935 None.

44936 **APPLICATION USAGE**

44937 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.

44938 **RATIONALE**

44939 None.

44940 **FUTURE DIRECTIONS**

44941 None.

44942 **SEE ALSO**

44943 *mmap()*, *sysconf()*

44944 XBD <[sys/mman.h](#)>

44945 **CHANGE HISTORY**

44946 First released in Issue 4, Version 2.

44947 **Issue 5**

44948 Moved from X/OPEN UNIX extension to BASE.

44949 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

44950 The DESCRIPTION is largely reworded.

44951 [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.

44952 [EAGAIN] is moved from the optional to the mandatory error conditions.

44953 **Issue 6**

44954 The *mprotect()* function is marked as part of the Memory Protection option.

44955 The following new requirements on POSIX implementations derive from alignment with the
44956 Single UNIX Specification:

44957 The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
44958 of the page size as returned by *sysconf()*.

44959 The [EINVAL] error condition is added.

44960 The normative text is updated to avoid use of the term “must” for application requirements.

44961 **Issue 7**

44962 SD5-XSH-ERN-22 is applied, deleting erroneous APPLICATION USAGE.

44963 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
44964 requirements.

44965 The *mprotect()* function is moved from the Memory Protection option to the Base.

44966 **NAME**

44967 mq_close ‡close a message queue (REALTIME)

44968 **SYNOPSIS**

```
44969 MSG #include <mqueue.h>
44970 int mq_close(mqd_t mqdes);
```

44971 **DESCRIPTION**

44972 The *mq_close()* function shall remove the association between the message queue descriptor,
 44973 *mqdes*, and its message queue. The results of using this message queue descriptor after successful
 44974 return from this *mq_close()*, and until the return of this message queue descriptor from a
 44975 subsequent *mq_open()*, are undefined.

44976 If the process has successfully attached a notification request to the message queue via this
 44977 *mqdes*, this attachment shall be removed, and the message queue is available for another process
 44978 to attach for notification.

44979 **RETURN VALUE**

44980 Upon successful completion, the *mq_close()* function shall return a value of zero; otherwise, the
 44981 function shall return a value of -1 and set *errno* to indicate the error.

44982 **ERRORS**44983 The *mq_close()* function shall fail if:

44984 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

44985 **EXAMPLES**

44986 None.

44987 **APPLICATION USAGE**

44988 None.

44989 **RATIONALE**

44990 None.

44991 **FUTURE DIRECTIONS**

44992 None.

44993 **SEE ALSO**44994 *mq_open()*, *mq_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*44995 XBD [<mqueue.h>](#)44996 **CHANGE HISTORY**

44997 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

44998 **Issue 6**44999 The *mq_close()* function is marked as part of the Message Passing option.

45000 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 45001 implementation does not support the Message Passing option.

45002 **NAME**45003 mq_getattr ‡get message queue attributes **REALTIME**)45004 **SYNOPSIS**

```
45005 MSG #include <mqueue.h>
45006 int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

45007 **DESCRIPTION**

45008 The *mq_getattr()* function shall obtain status information and attributes of the message queue
 45009 and the open message queue description associated with the message queue descriptor.

45010 The *mqdes* argument specifies a message queue descriptor.

45011 The results shall be returned in the **mq_attr** structure referenced by the *mqstat* argument.

45012 Upon return, the following members shall have the values associated with the open message
 45013 queue description as set when the message queue was opened and as modified by subsequent
 45014 *mq_setattr()* calls: *mq_flags*.

45015 The following attributes of the message queue shall be returned as set at message queue
 45016 creation: *mq_maxmsg*, *mq_msgsize*.

45017 Upon return, the following members within the **mq_attr** structure referenced by the *mqstat*
 45018 argument shall be set to the current state of the message queue:

45019 *mq_curmsgs* The number of messages currently on the queue.

45020 **RETURN VALUE**

45021 Upon successful completion, the *mq_getattr()* function shall return zero. Otherwise, the function
 45022 shall return -1 and set *errno* to indicate the error.

45023 **ERRORS**

45024 The *mq_getattr()* function may fail if:

45025 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

45026 **EXAMPLES**

45027 See *mq_notify()*.

45028 **APPLICATION USAGE**

45029 None.

45030 **RATIONALE**

45031 None.

45032 **FUTURE DIRECTIONS**

45033 None.

45034 **SEE ALSO**

45035 *mq_notify()*, *mq_open()*, *mq_send()*, *mq_setattr()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

45036 XBD <[mqueue.h](#)>

45037 **CHANGE HISTORY**

45038 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45039 **Issue 6**

45040 The *mq_getattr()* function is marked as part of the Message Passing option.

45041 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 45042 implementation does not support the Message Passing option.

45043
45044

The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

45045
45046

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/61 is applied, updating the ERRORS section to change the [EBADF] error from mandatory to optional.

45047 **NAME**45048 mq_notify — notify process that a message is available (**REALTIME**)45049 **SYNOPSIS**

```
45050 MSG #include <mqueue.h>
45051 int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

45052 **DESCRIPTION**

45053 If the argument *notification* is not NULL, this function shall register the calling process to be
 45054 notified of message arrival at an empty message queue associated with the specified message
 45055 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to
 45056 the process when the message queue transitions from empty to non-empty. At any time, only
 45057 one process may be registered for notification by a message queue. If the calling process or any
 45058 other process has already registered for notification of message arrival at the specified message
 45059 queue, subsequent attempts to register for that message queue shall fail.

45060 If *notification* is NULL and the process is currently registered for notification by the specified
 45061 message queue, the existing registration shall be removed.

45062 When the notification is sent to the registered process, its registration shall be removed. The
 45063 message queue shall then be available for registration.

45064 If a process has registered for notification of message arrival at a message queue and some
 45065 thread is blocked in *mq_receive()* or *mq_timedreceive()* waiting to receive a message when a
 45066 message arrives at the queue, the arriving message shall satisfy the appropriate *mq_receive()* or
 45067 *mq_timedreceive()*, respectively. The resulting behavior is as if the message queue remains empty,
 45068 and no notification shall be sent.

45069 **RETURN VALUE**

45070 Upon successful completion, the *mq_notify()* function shall return a value of zero; otherwise, the
 45071 function shall return a value of -1 and set *errno* to indicate the error.

45072 **ERRORS**

45073 The *mq_notify()* function shall fail if:

45074 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

45075 [EBUSY] A process is already registered for notification by the message queue.

45076 The *mq_notify()* function may fail if:

45077 [EINVAL] The *notification* argument is NULL and the process is currently not registered.

45078 **EXAMPLES**

45079 The following program registers a notification request for the message queue named in its
 45080 command-line argument. Notification is performed by creating a thread. The thread executes a
 45081 function which reads one message from the queue and then terminates the process.

```
45082 #include <pthread.h>
45083 #include <mqueue.h>
45084 #include <assert.h>
45085 #include <stdio.h>
45086 #include <stdlib.h>
45087 #include <unistd.h>

45088 static void /* Thread start function */
45089 tfunc(union sigval sv)
45090 {
```



```

45091     struct mq_attr attr;
45092     ssize_t nr;
45093     void *buf;
45094     mqd_t mqdes = *((mqd_t *) sv.sival_ptr);
45095
45096     /* Determine maximum msg size; allocate buffer to receive msg */
45097
45098     if (mq_getattr(mqdes, &attr) == -1) {
45099         perror("mq_getattr");
45100         exit(EXIT_FAILURE);
45101     }
45102     buf = malloc(attr.mq_msgsize);
45103
45104     if (buf == NULL) {
45105         perror("malloc");
45106         exit(EXIT_FAILURE);
45107     }
45108
45109     nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
45110     if (nr == -1) {
45111         perror("mq_receive");
45112         exit(EXIT_FAILURE);
45113     }
45114
45115     printf("Read %ld bytes from message queue\n", (long) nr);
45116     free(buf);
45117     exit(EXIT_SUCCESS);          /* Terminate the process */
45118 }
45119
45120 int
45121 main(int argc, char *argv[])
45122 {
45123     mqd_t mqdes;
45124     struct sigevent not;
45125
45126     assert(argc == 2);
45127
45128     mqdes = mq_open(argv[1], O_RDONLY);
45129     if (mqdes == (mqd_t) -1) {
45130         perror("mq_open");
45131         exit(EXIT_FAILURE);
45132     }
45133
45134     not.sigev_notify = SIGEV_THREAD;
45135     not.sigev_notify_function = tfunc;
45136     not.sigev_notify_attributes = NULL;
45137     not.sigev_value.sival_ptr = &mqdes;    /* Arg. to thread func. */
45138     if (mq_notify(mqdes, &not) == -1) {
45139         perror("mq_notify");
45140         exit(EXIT_FAILURE);
45141     }
45142
45143     pause();    /* Process will be terminated by thread function */
45144 }

```

45135 APPLICATION USAGE

45136 None.

45137 RATIONALE

45138 None.

45139 FUTURE DIRECTIONS

45140 None.

45141 SEE ALSO

45142 *mq_open()*, *mq_send()*, *mq_receive()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

45143 XBD <[mqqueue.h](#)>

45144 CHANGE HISTORY

45145 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45146 Issue 6

45147 The *mq_notify()* function is marked as part of the Message Passing option.

45148 The [ENOSYS] error condition has been removed as stubs need not be provided if an
45149 implementation does not support the Message Passing option.

45150 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std
45151 1003.1d-1999.

45152 Issue 7

45153 SD5-XSH-ERN-38 is applied, adding the *mq_timedreceive()* function to the DESCRIPTION.

45154 Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.

45155 An example is added.

45156 **NAME**

45157 mq_open ‡open a message queue (REALTIME)

45158 **SYNOPSIS**

```
45159 MSG #include <mqueue.h>
45160 mqd_t mq_open(const char *name, int oflag, ...);
```

45161 **DESCRIPTION**

45162 The *mq_open()* function shall establish the connection between a process and a message queue
 45163 with a message queue descriptor. It shall create an open message queue description that refers to
 45164 the message queue, and a message queue descriptor that refers to that open message queue
 45165 description. The message queue descriptor is used by other functions to refer to that message
 45166 queue. The *name* argument points to a string naming a message queue. It is unspecified whether
 45167 the name appears in the file system and is visible to other functions that take pathnames as
 45168 arguments. The *name* argument conforms to the construction rules for a pathname, except that
 45169 the interpretation of <slash> characters other than the leading <slash> character in *name* is
 45170 implementation-defined, and that the length limits for the *name* argument are implementation-
 45171 defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If
 45172 *name* begins with the <slash> character, then processes calling *mq_open()* with the same value of
 45173 *name* shall refer to the same message queue object, as long as that name has not been removed. If
 45174 *name* does not begin with the <slash> character, the effect is implementation-defined. If the *name*
 45175 argument is not the name of an existing message queue and creation is not requested, *mq_open()*
 45176 shall fail and return an error.

45177 A message queue descriptor may be implemented using a file descriptor, in which case
 45178 applications can open up to at least {OPEN_MAX} file and message queues.

45179 The *oflag* argument requests the desired receive and/or send access to the message queue. The
 45180 requested access permission to receive messages or send messages shall be granted if the calling
 45181 process would be granted read or write access, respectively, to an equivalently protected file.

45182 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications
 45183 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

45184 **O_RDONLY** Open the message queue for receiving messages. The process can use the
 45185 returned message queue descriptor with *mq_receive()*, but not *mq_send()*. A
 45186 message queue may be open multiple times in the same or different processes
 45187 for receiving messages.

45188 **O_WRONLY** Open the queue for sending messages. The process can use the returned
 45189 message queue descriptor with *mq_send()* but not *mq_receive()*. A message
 45190 queue may be open multiple times in the same or different processes for
 45191 sending messages.

45192 **O_RDWR** Open the queue for both receiving and sending messages. The process can use
 45193 any of the functions allowed for **O_RDONLY** and **O_WRONLY**. A message
 45194 queue may be open multiple times in the same or different processes for
 45195 sending messages.

45196 Any combination of the remaining flags may be specified in the value of *oflag*:

45197 **O_CREAT** Create a message queue. It requires two additional arguments: *mode*, which
 45198 shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr**
 45199 structure. If the pathname *name* has already been used to create a message
 45200 queue that still exists, then this flag shall have no effect, except as noted under
 45201 **O_EXCL**. Otherwise, a message queue shall be created without any messages

45202 in it. The user ID of the message queue shall be set to the effective user ID of
 45203 the process. The group ID of the message queue shall be set to the effective
 45204 group ID of the process; however, if the *name* argument is visible in the file
 45205 system, the group ID may be set to the group ID of the containing directory.
 45206 When bits in *mode* other than the file permission bits are specified, the effect is
 45207 unspecified. If *attr* is NULL, the message queue shall be created with
 45208 implementation-defined default message queue attributes. If *attr* is non-NULL
 45209 and the calling process has appropriate privileges on *name*, the message queue
 45210 *mq_maxmsg* and *mq_msgsize* attributes shall be set to the values of the
 45211 corresponding members in the **mq_attr** structure referred to by *attr*. The
 45212 values of the *mq_flags* and *mq_curmsgs* members of the **mq_attr** structure shall
 45213 be ignored. If *attr* is non-NULL, but the calling process does not have
 45214 appropriate privileges on *name*, the *mq_open()* function shall fail and return an
 45215 error without creating the message queue.

45216 **O_EXCL** If O_EXCL and O_CREAT are set, *mq_open()* shall fail if the message queue
 45217 *name* exists. The check for the existence of the message queue and the creation
 45218 of the message queue if it does not exist shall be atomic with respect to other
 45219 threads executing *mq_open()* naming the same *name* with O_EXCL and
 45220 O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is
 45221 undefined.

45222 **O_NONBLOCK** Determines whether an *mq_send()* or *mq_receive()* waits for resources or
 45223 messages that are not currently available, or fails with *errno* set to [EAGAIN];
 45224 see *mq_send()* and *mq_receive()* for details.

45225 The *mq_open()* function does not add or remove messages from the queue.

45226 RETURN VALUE

45227 Upon successful completion, the function shall return a message queue descriptor; otherwise,
 45228 the function shall return (**mqd_t**)-1 and set *errno* to indicate the error.

45229 ERRORS

45230 The *mq_open()* function shall fail if:

45231 [EACCES] The message queue exists and the permissions specified by *oflag* are denied, or
 45232 the message queue does not exist and permission to create the message queue
 45233 is denied.

45234 [EEXIST] O_CREAT and O_EXCL are set and the named message queue already exists.

45235 [EINTR] The *mq_open()* function was interrupted by a signal.

45236 [EINVAL] The *mq_open()* function is not supported for the given name.

45237 [EINVAL] O_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either
 45238 *mq_maxmsg* or *mq_msgsize* was less than or equal to zero.

45239 [EMFILE] Too many message queue descriptors or file descriptors are currently in use by
 45240 this process.

45241 [ENFILE] Too many message queues are currently open in the system.

45242 [ENOENT] O_CREAT is not set and the named message queue does not exist.

45243 [ENOSPC] There is insufficient space for the creation of the new message queue.

45244 If any of the following conditions occur, the *mq_open()* function may return (**mqd_t**)-1 and set
 45245 *errno* to the corresponding value.

45246 [ENAMETOOLONG]

45247 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 45248 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 45249 systems, or has a pathname component that is longer than
 45250 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 45251 longer than {_XOPEN_NAME_MAX} on XSI systems.

45252 EXAMPLES

45253 None.

45254 APPLICATION USAGE

45255 None.

45256 RATIONALE

45257 None.

45258 FUTURE DIRECTIONS

45259 A future version might require the *mq_open()* and *mq_unlink()* functions to have semantics
 45260 similar to normal file system operations.

45261 SEE ALSO

45262 *mq_close()*, *mq_getattr()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgctl()*, *msgget()*,
 45263 *msgrcv()*, *msgsnd()*

45264 XBD <[mqqueue.h](#)>

45265 CHANGE HISTORY

45266 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45267 Issue 6

45268 The *mq_open()* function is marked as part of the Message Passing option.

45269 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 45270 implementation does not support the Message Passing option.

45271 The *mq_timedreceive()* and *mq_timedsend()* functions are added to the SEE ALSO section for
 45272 alignment with IEEE Std 1003.1d-1999.

45273 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

45274 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/62 is applied, updating the description of
 45275 the permission bits in the DESCRIPTION. The change is made for consistency with the
 45276 *shm_open()* and *sem_open()* functions.

45277 Issue 7

45278 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and
 45279 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

45280 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

45281 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the
 45282 user ID and group ID of the message queue.

45283 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0394 [259] is applied.

45284 **NAME**45285 mq_receive, mq_timedreceive — receive a message from a message queue (**REALTIME**)45286 **SYNOPSIS**

```

45287 MSG #include <mqueue.h>
45288      ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
45289                       unsigned *msg_prio);
45290
45291 #include <mqueue.h>
45292 #include <time.h>
45293
45294      ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
45295                             size_t msg_len, unsigned *restrict msg_prio,
45296                             const struct timespec *restrict abstime);

```

45295 **DESCRIPTION**

45296 The *mq_receive()* function shall receive the oldest of the highest priority message(s) from the
 45297 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len*
 45298 argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and
 45299 return an error. Otherwise, the selected message shall be removed from the queue and copied to
 45300 the buffer pointed to by the *msg_ptr* argument.

45301 If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.

45302 If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the
 45303 location referenced by *msg_prio*.

45304 If the specified message queue is empty and O_NONBLOCK is not set in the message queue
 45305 description associated with *mqdes*, *mq_receive()* shall block until a message is enqueued on the
 45306 message queue or until *mq_receive()* is interrupted by a signal. If more than one thread is waiting
 45307 to receive a message when a message arrives at an empty queue and the Priority Scheduling
 45308 option is supported, then the thread of highest priority that has been waiting the longest shall be
 45309 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the
 45310 message. If the specified message queue is empty and O_NONBLOCK is set in the message
 45311 queue description associated with *mqdes*, no message shall be removed from the queue, and
 45312 *mq_receive()* shall return an error.

45313 The *mq_timedreceive()* function shall receive the oldest of the highest priority messages from the
 45314 message queue specified by *mqdes* as described for the *mq_receive()* function. However, if
 45315 O_NONBLOCK was not specified when the message queue was opened via the *mq_open()*
 45316 function, and no message exists on the queue to satisfy the receive, the wait for such a message
 45317 shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is
 45318 equivalent to *mq_receive()*.

45319 The timeout expires when the absolute time specified by *abstime* passes, as measured by the
 45320 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 45321 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
 45322 call.

45323 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 45324 be the resolution of the clock on which it is based. The *timespec* argument is defined in the
 45325 **<time.h>** header.

45326 Under no circumstance shall the operation fail with a timeout if a message can be removed from
 45327 the message queue immediately. The validity of the *abstime* parameter need not be checked if a
 45328 message can be removed from the message queue immediately.

45329 **RETURN VALUE**

45330 Upon successful completion, the *mq_receive()* and *mq_timedreceive()* functions shall return the
 45331 length of the selected message in bytes and the message shall be removed from the queue.
 45332 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1 ,
 45333 and set *errno* to indicate the error.

45334 **ERRORS**

45335 These functions shall fail if:

45336 [EAGAIN] O_NONBLOCK was set in the message description associated with *mqdes*, and
 45337 the specified message queue is empty.

45338 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.

45339 [EMSGSIZE] The specified message buffer size, *msg_len*, is less than the message size
 45340 attribute of the message queue.

45341 [EINTR] The *mq_receive()* or *mq_timedreceive()* operation was interrupted by a signal.

45342 [EINVAL] The process or thread would have blocked, and the *abstime* parameter
 45343 specified a nanoseconds field value less than zero or greater than or equal to
 45344 1 000 million.

45345 [ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened,
 45346 but no message arrived on the queue before the specified timeout expired.

45347 These functions may fail if:

45348 [EBADMSG] The implementation has detected a data corruption problem with the
 45349 message.

45350 **EXAMPLES**

45351 None.

45352 **APPLICATION USAGE**

45353 None.

45354 **RATIONALE**

45355 None.

45356 **FUTURE DIRECTIONS**

45357 None.

45358 **SEE ALSO**

45359 [*mq_open\(\)*](#), [*mq_send\(\)*](#), [*msgctl\(\)*](#), [*msgget\(\)*](#), [*msgrcv\(\)*](#), [*msgsnd\(\)*](#), [*time\(\)*](#)

45360 XBD [**<mqqueue.h>**](#), [**<time.h>**](#)

45361 **CHANGE HISTORY**

45362 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45363 **Issue 6**

45364 The *mq_receive()* function is marked as part of the Message Passing option.

45365 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to
 45366 *msg_len* rather than *maxsize*.

45367 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 45368 implementation does not support the Message Passing option.

45369 The following new requirements on POSIX implementations derive from alignment with the
 45370 Single UNIX Specification:

45371 In this function it is possible for the return value to exceed the range of the type `ssize_t`
45372 (since `size_t` has a larger range of positive values than `ssize_t`). A sentence restricting the
45373 size of the `size_t` object is added to the description to resolve this conflict.

45374 The `mq_timedreceive()` function is added for alignment with IEEE Std 1003.1d-1999.

45375 The `restrict` keyword is added to the `mq_timedreceive()` prototype for alignment with the
45376 ISO/IEC 9899:1999 standard.

45377 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for `mq_timedreceive()`
45378 from `int` to `ssize_t`.

45379 **Issue 7**

45380 The `mq_timedreceive()` function is moved from the Timeouts option to the Base.

45381 Functionality relating to the Timers option is moved to the Base.

45382 **NAME**45383 mq_send, mq_timedsend ‡send a message to a message queue **REALTIME**)45384 **SYNOPSIS**

```

45385 MSG #include <mqueue.h>
45386 int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
45387             unsigned msg_prio);
45388 #include <mqueue.h>
45389 #include <time.h>
45390 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
45391                 unsigned msg_prio, const struct timespec *abstime);

```

45392 **DESCRIPTION**

45393 The *mq_send()* function shall add the message pointed to by the argument *msg_ptr* to the
 45394 message queue specified by *mqdes*. The *msg_len* argument specifies the length of the message, in
 45395 bytes, pointed to by *msg_ptr*. The value of *msg_len* shall be less than or equal to the *mq_msgsize*
 45396 attribute of the message queue, or *mq_send()* shall fail.

45397 If the specified message queue is not full, *mq_send()* shall behave as if the message is inserted
 45398 into the message queue at the position indicated by the *msg_prio* argument. A message with a
 45399 larger numeric value of *msg_prio* shall be inserted before messages with lower values of
 45400 *msg_prio*. A message shall be inserted after other messages in the queue, if any, with equal
 45401 *msg_prio*. The value of *msg_prio* shall be less than {MQ_PRIO_MAX}.

45402 If the specified message queue is full and O_NONBLOCK is not set in the message queue
 45403 description associated with *mqdes*, *mq_send()* shall block until space becomes available to
 45404 enqueue the message, or until *mq_send()* is interrupted by a signal. If more than one thread is
 45405 waiting to send when space becomes available in the message queue and the Priority Scheduling
 45406 option is supported, then the thread of the highest priority that has been waiting the longest
 45407 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is
 45408 unblocked. If the specified message queue is full and O_NONBLOCK is set in the message
 45409 queue description associated with *mqdes*, the message shall not be queued and *mq_send()* shall
 45410 return an error.

45411 The *mq_timedsend()* function shall add a message to the message queue specified by *mqdes* in the
 45412 manner defined for the *mq_send()* function. However, if the specified message queue is full and
 45413 O_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for
 45414 sufficient room in the queue shall be terminated when the specified timeout expires. If
 45415 O_NONBLOCK is set in the message queue description, this function shall be equivalent to
 45416 *mq_send()*.

45417 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
 45418 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 45419 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
 45420 call.

45421 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 45422 be the resolution of the clock on which it is based. The *timespec* argument is defined in the
 45423 **<time.h>** header.

45424 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the
 45425 queue to add the message immediately. The validity of the *abstime* parameter need not be
 45426 checked when there is sufficient room in the queue.

45427 **RETURN VALUE**

45428 Upon successful completion, the `mq_send()` and `mq_timedsend()` functions shall return a value of
 45429 zero. Otherwise, no message shall be enqueued, the functions shall return `-1`, and `errno` shall be
 45430 set to indicate the error.

45431 **ERRORS**

45432 The `mq_send()` and `mq_timedsend()` functions shall fail if:

45433 [EAGAIN] The `O_NONBLOCK` flag is set in the message queue description associated
 45434 with `mqdes`, and the specified message queue is full.

45435 [EBADF] The `mqdes` argument is not a valid message queue descriptor open for writing.

45436 [EINTR] A signal interrupted the call to `mq_send()` or `mq_timedsend()`.

45437 [EINVAL] The value of `msg_prio` was outside the valid range.

45438 [EINVAL] The process or thread would have blocked, and the `abstime` parameter
 45439 specified a nanoseconds field value less than zero or greater than or equal to
 45440 1 000 million.

45441 [EMSGSIZE] The specified message length, `msg_len`, exceeds the message size attribute of
 45442 the message queue.

45443 [ETIMEDOUT] The `O_NONBLOCK` flag was not set when the message queue was opened,
 45444 but the timeout expired before the message could be added to the queue.

45445 **EXAMPLES**

45446 None.

45447 **APPLICATION USAGE**

45448 The value of the symbol `{MQ_PRIO_MAX}` limits the number of priority levels supported by the
 45449 application. Message priorities range from 0 to `{MQ_PRIO_MAX}-1`.

45450 **RATIONALE**

45451 None.

45452 **FUTURE DIRECTIONS**

45453 None.

45454 **SEE ALSO**

45455 [`mq_open\(\)`](#), [`mq_receive\(\)`](#), [`mq_setattr\(\)`](#), [`time\(\)`](#)

45456 XBD [`<mqueue.h>`](#), [`<time.h>`](#)

45457 **CHANGE HISTORY**

45458 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45459 **Issue 6**

45460 The `mq_send()` function is marked as part of the Message Passing option.

45461 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 45462 implementation does not support the Message Passing option.

45463 The `mq_timedsend()` function is added for alignment with IEEE Std 1003.1d-1999.

45464 **Issue 7**

45465 The `mq_timedsend()` function is moved from the Timeouts option to the Base.

45466 Functionality relating to the Timers option is moved to the Base.

45467 **NAME**45468 mq_setattr ‡'set message queue attributes **REALTIME**)45469 **SYNOPSIS**

```
45470 MSG #include <mqueue.h>
45471 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
45472 struct mq_attr *restrict omqstat);
```

45473 **DESCRIPTION**

45474 The *mq_setattr()* function shall set attributes associated with the open message queue
 45475 description referenced by the message queue descriptor specified by *mqdes*.

45476 The message queue attributes corresponding to the following members defined in the **mq_attr**
 45477 structure shall be set to the specified values upon successful completion of *mq_setattr()*:

45478 *mq_flags* The value of this member is the bitwise-logical OR of zero or more of
 45479 **O_NONBLOCK** and any implementation-defined flags.

45480 The values of the *mq_maxmsg*, *mq_msgsize*, and *mq_curmsgs* members of the **mq_attr** structure
 45481 shall be ignored by *mq_setattr()*.

45482 If *omqstat* is non-NULL, the *mq_setattr()* function shall store, in the location referenced by
 45483 *omqstat*, the previous message queue attributes and the current queue status. These values shall
 45484 be the same as would be returned by a call to *mq_getattr()* at that point.

45485 **RETURN VALUE**

45486 Upon successful completion, the function shall return a value of zero and the attributes of the
 45487 message queue shall have been changed as specified.

45488 Otherwise, the message queue attributes shall be unchanged, and the function shall return a
 45489 value of -1 and set *errno* to indicate the error.

45490 **ERRORS**

45491 The *mq_setattr()* function shall fail if:

45492 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

45493 **EXAMPLES**

45494 None.

45495 **APPLICATION USAGE**

45496 None.

45497 **RATIONALE**

45498 None.

45499 **FUTURE DIRECTIONS**

45500 None.

45501 **SEE ALSO**

45502 *mq_open()*, *mq_send()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

45503 XBD [<mqueue.h>](#)

45504 **CHANGE HISTORY**

45505 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45506 **Issue 6**

45507 The *mq_setattr()* function is marked as part of the Message Passing option.

45508 The [ENOSYS] error condition has been removed as stubs need not be provided if an
45509 implementation does not support the Message Passing option.

45510 The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std
45511 1003.1d-1999.

45512 The **restrict** keyword is added to the *mq_setattr()* prototype for alignment with the
45513 ISO/IEC 9899:1999 standard.

45514 **NAME**45515 mq_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)45516 **SYNOPSIS**

```
45517 MSG       #include <mqueue.h>
45518           #include <time.h>
45519           ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
45520                                   size_t msg_len, unsigned *restrict msg_prio,
45521                                   const struct timespec *restrict abstime);
```

45522 **DESCRIPTION**45523 Refer to [mq_receive\(\)](#).

45524 **NAME**

45525 mq_timedsend †'send a message to a message queue (ADVANCED REALTIME)

45526 **SYNOPSIS**

```
45527 MSG #include <mqueue.h>
45528 #include <time.h>
45529 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
45530 unsigned msg_prio, const struct timespec *abstime);
```

45531 **DESCRIPTION**45532 Refer to *mq_send()*.

45533 **NAME**45534 mq_unlink — remove a message queue (**REALTIME**)45535 **SYNOPSIS**

```
45536 MSG #include <mqueue.h>
45537 int mq_unlink(const char *name);
```

45538 **DESCRIPTION**

45539 The *mq_unlink()* function shall remove the message queue named by the string *name*. If one or
 45540 more processes have the message queue open when *mq_unlink()* is called, destruction of the
 45541 message queue shall be postponed until all references to the message queue have been closed.
 45542 However, the *mq_unlink()* call need not block until all references have been closed; it may return
 45543 immediately.

45544 After a successful call to *mq_unlink()*, reuse of the name shall subsequently cause *mq_open()* to
 45545 behave as if no message queue of this name exists (that is, *mq_open()* will fail if *O_CREAT* is not
 45546 set, or will create a new message queue if *O_CREAT* is set).

45547 **RETURN VALUE**

45548 Upon successful completion, the function shall return a value of zero. Otherwise, the named
 45549 message queue shall be unchanged by this function call, and the function shall return a value of
 45550 -1 and set *errno* to indicate the error.

45551 **ERRORS**45552 The *mq_unlink()* function shall fail if:

- 45553 [EACCES] Permission is denied to unlink the named message queue.
- 45554 [EINTR] The call to *mq_unlink()* blocked waiting for all references to the named
 45555 message queue to be closed and a signal interrupted the call.
- 45556 [ENOENT] The named message queue does not exist.

45557 The *mq_unlink()* function may fail if:

- 45558 [ENAMETOOLONG]
- 45559 The length of the *name* argument exceeds $\{_POSIX_PATH_MAX\}$ on systems
 45560 XSI that do not support the XSI option or exceeds $\{_XOPEN_PATH_MAX\}$ on XSI
 45561 systems, or has a pathname component that is longer than
 45562 XSI $\{_POSIX_NAME_MAX\}$ on systems that do not support the XSI option or
 45563 longer than $\{_XOPEN_NAME_MAX\}$ on XSI systems. A call to *mq_unlink()*
 45564 with a *name* argument that contains the same message queue name as was
 45565 previously used in a successful *mq_open()* call shall not give an
 45566 [ENAMETOOLONG] error.

45567 **EXAMPLES**

45568 None.

45569 **APPLICATION USAGE**

45570 None.

45571 **RATIONALE**

45572 None.

45573 **FUTURE DIRECTIONS**

45574 A future version might require the *mq_open()* and *mq_unlink()* functions to have semantics
45575 similar to normal file system operations.

45576 **SEE ALSO**

45577 *mq_close()*, *mq_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

45578 XBD <[mqqueue.h](#)>

45579 **CHANGE HISTORY**

45580 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

45581 **Issue 6**

45582 The *mq_unlink()* function is marked as part of the Message Passing option.

45583 The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion,
45584 the named message queue is unchanged by this function.

45585 The [ENOSYS] error condition has been removed as stubs need not be provided if an
45586 implementation does not support the Message Passing option.

45587 **Issue 7**

45588 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
45589 ``shall fail'' to a ``may fail'' error .

45590 Austin Group Interpretation 1003.1-2001 #141 is applied.

45591 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0230 [504] is applied.

45592 **NAME**

45593 mrnd48 ‡generate uniformly distributed pseudo-random signed long integers

45594 **SYNOPSIS**

```
45595 XSI       #include <stdlib.h>  
45596       long mrnd48(void);
```

45597 **DESCRIPTION**

45598 Refer to *drand48()*.

45599 **NAME**

45600 msgctl — XSI message control operations

45601 **SYNOPSIS**

```
45602 XSI #include <sys/msg.h>
45603 int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

45604 **DESCRIPTION**

45605 The *msgctl()* function operates on XSI message queues (see XBD [Section 3.226](#), on page 69). It is
 45606 unspecified whether this function interoperates with the realtime interprocess communication
 45607 facilities defined in [Section 2.8](#) (on page 503).

45608 The *msgctl()* function shall provide message control operations as specified by *cmd*. The
 45609 following values for *cmd*, and the message control operations they specify, are:

45610 **IPC_STAT** Place the current value of each member of the **msqid_ds** data structure
 45611 associated with *msqid* into the structure pointed to by *buf*. The contents of this
 45612 structure are defined in **<sys/msg.h>**.

45613 **IPC_SET** Set the value of the following members of the **msqid_ds** data structure
 45614 associated with *msqid* to the corresponding value found in the structure
 45615 pointed to by *buf*:

```
45616 msg_perm.uid
45617 msg_perm.gid
45618 msg_perm.mode
45619 msg_qbytes
```

45620 Also, the *msg_ctime* timestamp shall be set to the current time, as described in
 45621 [Section 2.7.1](#) (on page 502).

45622 **IPC_SET** can only be executed by a process with appropriate privileges or that
 45623 has an effective user ID equal to the value of **msg_perm.cuid** or
 45624 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a
 45625 process with appropriate privileges can raise the value of **msg_qbytes**.

45626 **IPC_RMID** Remove the message queue identifier specified by *msqid* from the system and
 45627 destroy the message queue and **msqid_ds** data structure associated with it.
 45628 **IPC_RMID** can only be executed by a process with appropriate privileges or
 45629 one that has an effective user ID equal to the value of **msg_perm.cuid** or
 45630 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

45631 **RETURN VALUE**

45632 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to
 45633 indicate the error.

45634 **ERRORS**

45635 The *msgctl()* function shall fail if:

45636 **[EACCES]** The argument *cmd* is **IPC_STAT** and the calling process does not have read
 45637 permission; see [Section 2.7](#) (on page 501).

45638 **[EINVAL]** The value of *msqid* is not a valid message queue identifier; or the value of *cmd*
 45639 is not a valid command.

45640 **[EPERM]** The argument *cmd* is **IPC_RMID** or **IPC_SET** and the effective user ID of the
 45641 calling process is not equal to that of a process with appropriate privileges and
 45642 it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data

45643 structure associated with *msqid*.

45644 [EPERM] The argument *cmd* is `IPC_SET`, an attempt is being made to increase to the
45645 value of `msg_qbytes`, and the effective user ID of the calling process does not
45646 have appropriate privileges.

45647 **EXAMPLES**

45648 None.

45649 **APPLICATION USAGE**

45650 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
45651 (IPC). Application developers who need to use IPC should design their applications so that
45652 modules using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to
45653 use the alternative interfaces.

45654 **RATIONALE**

45655 None.

45656 **FUTURE DIRECTIONS**

45657 None.

45658 **SEE ALSO**

45659 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
45660 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

45661 [XBD Section 3.226](#) (on page 69), [<sys/msg.h>](#)

45662 **CHANGE HISTORY**

45663 First released in Issue 2. Derived from Issue 2 of the SVID.

45664 **Issue 5**

45665 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
45666 DIRECTIONS to a new APPLICATION USAGE section.

45667 **Issue 7**

45668 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0395 [345] is applied.

45669 **NAME**45670 `msgget` ‡get the XSI message queue identifier45671 **SYNOPSIS**

```
45672 XSI #include <sys/msg.h>
45673 int msgget(key_t key, int msgflg);
```

45674 **DESCRIPTION**

45675 The `msgget()` function operates on XSI message queues (see XBD [Section 3.226](#), on page 69). It is
 45676 unspecified whether this function interoperates with the realtime interprocess communication
 45677 facilities defined in [Section 2.8](#) (on page 503).

45678 The `msgget()` function shall return the message queue identifier associated with the argument
 45679 `key`.

45680 A message queue identifier, associated message queue, and data structure (see `<sys/msg.h>`),
 45681 shall be created for the argument `key` if one of the following is true:

45682 The argument `key` is equal to `IPC_PRIVATE`.

45683 The argument `key` does not already have a message queue identifier associated with it, and
 45684 `(msgflg & IPC_CREAT)` is non-zero.

45685 Upon creation, the data structure associated with the new message queue identifier shall be
 45686 initialized as follows:

45687 `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` shall be set to the
 45688 effective user ID and effective group ID, respectively, of the calling process.

45689 The low-order 9 bits of `msg_perm.mode` shall be set to the low-order 9 bits of `msgflg`.

45690 `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` shall be set to 0.

45691 `msg_ctime` shall be set to the current time, as described in [Section 2.7.1](#) (on page 502).

45692 `msg_qbytes` shall be set to the system limit.

45693 **RETURN VALUE**

45694 Upon successful completion, `msgget()` shall return a non-negative integer, namely a message
 45695 queue identifier. Otherwise, it shall return `-1` and set `errno` to indicate the error.

45696 **ERRORS**

45697 The `msgget()` function shall fail if:

45698 [EACCES] A message queue identifier exists for the argument `key`, but operation
 45699 permission as specified by the low-order 9 bits of `msgflg` would not be granted;
 45700 see [Section 2.7](#) (on page 501).

45701 [EEXIST] A message queue identifier exists for the argument `key` but `((msgflg &
 45702 IPC_CREAT) && (msgflg & IPC_EXCL))` is non-zero.

45703 [ENOENT] A message queue identifier does not exist for the argument `key` and `(msgflg &
 45704 IPC_CREAT)` is 0.

45705 [ENOSPC] A message queue identifier is to be created but the system-imposed limit on
 45706 the maximum number of allowed message queue identifiers system-wide
 45707 would be exceeded.

45708 **EXAMPLES**

45709 None.

45710 **APPLICATION USAGE**

45711 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
45712 (IPC). Application developers who need to use IPC should design their applications so that
45713 modules using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to
45714 use the alternative interfaces.

45715 **RATIONALE**

45716 None.

45717 **FUTURE DIRECTIONS**

45718 None.

45719 **SEE ALSO**

45720 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), [ftok\(\)](#), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
45721 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

45722 [XBD Section 3.226](#) (on page 69), [<sys/msg.h>](#)45723 **CHANGE HISTORY**

45724 First released in Issue 2. Derived from Issue 2 of the SVID.

45725 **Issue 5**

45726 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
45727 DIRECTIONS to a new APPLICATION USAGE section.

45728 **Issue 7**

45729 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0396 [345] and XSH/TC1-2008/0397
45730 [344] are applied.

45731 **NAME**

45732 msgrcv — XSI message receive operation

45733 **SYNOPSIS**

```
45734 XSI #include <sys/msg.h>
45735 ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
45736 int msgflg);
```

45737 **DESCRIPTION**

45738 The *msgrcv()* function operates on XSI message queues (see XBD Section 3.226, on page 69). It is
 45739 unspecified whether this function interoperates with the realtime interprocess communication
 45740 facilities defined in Section 2.8 (on page 503).

45741 The *msgrcv()* function shall read a message from the queue associated with the message queue
 45742 identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

45743 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains
 45744 first a field of type **long** specifying the type of the message, and then a data portion that holds
 45745 the data bytes of the message. The structure below is an example of what this user-defined
 45746 buffer might look like:

```
45747 struct mymsg {
45748     long    mtype;    /* Message type. */
45749     char    mtext[1]; /* Message text. */
45750 }
```

45751 The structure member *mtype* is the received message's type as specified by the sending process.

45752 The structure member *mtext* is the text of the message.

45753 The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated
 45754 to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The
 45755 truncated part of the message shall be lost and no indication of the truncation shall be given to
 45756 the calling process.

45757 If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

45758 The argument *msgtyp* specifies the type of message requested as follows:

45759 If *msgtyp* is 0, the first message on the queue shall be received.

45760 If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.

45761 If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the
 45762 absolute value of *msgtyp* shall be received.

45763 The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the
 45764 queue. These are as follows:

45765 If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a
 45766 return value of -1 and *errno* set to [ENOMSG].

45767 If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the
 45768 following occurs:

45769 † A message of the desired type is placed on the queue.

45770 † The message queue identifier *msqid* is removed from the system; when this occurs,
 45771 *errno* shall be set to [EIDRM] and -1 shall be returned.

45772 ‡ If the calling thread receives a signal that is to be caught; in this case a message is not
 45773 received and the calling thread resumes execution in the manner prescribed in
 45774 *sigaction()*.

45775 Upon successful completion, the following actions are taken with respect to the data structure
 45776 associated with *msqid*:

45777 **msg_qnum** shall be decremented by 1.

45778 **msg_lrpid** shall be set to the process ID of the calling process.

45779 **msg_rtime** shall be set to the current time, as described in [Section 2.7.1](#) (on page 502).

45780 RETURN VALUE

45781 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually
 45782 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return -1 ,
 45783 and *errno* shall be set to indicate the error.

45784 ERRORS

45785 The *msgrcv()* function shall fail if:

45786 [E2BIG] The value of *mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is 0.

45787 [EACCES] Operation permission is denied to the calling process; see [Section 2.7](#) (on page
 45788 501).

45789 [EIDRM] The message queue identifier *msqid* is removed from the system.

45790 [EINTR] The *msgrcv()* function was interrupted by a signal.

45791 [EINVAL] *msqid* is not a valid message queue identifier.

45792 [ENOMSG] The queue does not contain a message of the desired type and (*msgflg* &
 45793 IPC_NOWAIT) is non-zero.

45794 EXAMPLES

45795 Receiving a Message

45796 The following example receives the first message on the queue (based on the value of the *msgtyp*
 45797 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has
 45798 previously been set). This call specifies that an error should be reported if no message is
 45799 available, but not if the message is too large. The message size is calculated directly using the
 45800 *sizeof* operator.

```

45801 #include <sys/msg.h>
45802 ...
45803 int result;
45804 int msqid;
45805 struct message {
45806     long type;
45807     char text[20];
45808 } msg;
45809 long msgtyp = 0;
45810 ...
45811 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
45812               msgtyp, MSG_NOERROR | IPC_NOWAIT);
  
```

45813 APPLICATION USAGE

45814 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
45815 (IPC). Application developers who need to use IPC should design their applications so that
45816 modules using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to
45817 use the alternative interfaces.

45818 RATIONALE

45819 None.

45820 FUTURE DIRECTIONS

45821 None.

45822 SEE ALSO

45823 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
45824 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgsnd\(\)](#),
45825 [sigaction\(\)](#)

45826 XBD [Section 3.226](#) (on page 69), [<sys/msg.h>](#)

45827 CHANGE HISTORY

45828 First released in Issue 2. Derived from Issue 2 of the SVID.

45829 Issue 5

45830 The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the
45831 DESCRIPTION about values of *msgsz* larger than {SSIZE_MAX}.

45832 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
45833 DIRECTIONS to the APPLICATION USAGE section.

45834 Issue 6

45835 The normative text is updated to avoid use of the term “must” for application requirements.

45836 Issue 7

45837 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0398 [345] and XSH/TC1-2008/0399
45838 [421] are applied.

45839 **NAME**

45840 msgsnd ‡XSI message send operation

45841 **SYNOPSIS**

```
45842 XSI #include <sys/msg.h>
45843 int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

45844 **DESCRIPTION**

45845 The *msgsnd()* function operates on XSI message queues (see XBD [Section 3.226](#), on page 69). It is
 45846 unspecified whether this function interoperates with the realtime interprocess communication
 45847 facilities defined in [Section 2.8](#) (on page 503).

45848 The *msgsnd()* function shall send a message to the queue associated with the message queue
 45849 identifier specified by *msqid*.

45850 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains
 45851 first a field of type **long** specifying the type of the message, and then a data portion that holds
 45852 the data bytes of the message. The structure below is an example of what this user-defined
 45853 buffer might look like:

```
45854 struct mymsg {
45855     long    mtype;        /* Message type. */
45856     char    mtext[1];    /* Message text. */
45857 }
```

45858 The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving
 45859 process for message selection.

45860 The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range
 45861 from 0 to a system-imposed maximum.

45862 The argument *msgflg* specifies the action to be taken if one or more of the following is true:

45863 ‡ If the number of bytes already on the queue is equal to **msg_qbytes**; see [<sys/msg.h>](#).

45864 ‡ If the total number of messages on all queues system-wide is equal to the system-imposed
 45865 limit.

45866 These actions are as follows:

45867 ‡ If (*msgflg* & IPC_NOWAIT) is non-zero, the message shall not be sent and the calling
 45868 thread shall return immediately.

45869 ‡ If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the
 45870 following occurs:

45871 ‡ If the condition responsible for the suspension no longer exists, in which case the
 45872 message is sent.

45873 ‡ If the message queue identifier *msqid* is removed from the system; when this occurs,
 45874 *errno* shall be set to [EIDRM] and -1 shall be returned.

45875 ‡ If the calling thread receives a signal that is to be caught; in this case the message is not
 45876 sent and the calling thread resumes execution in the manner prescribed in [sigaction\(\)](#).

45877 Upon successful completion, the following actions are taken with respect to the data structure
45878 associated with *msqid*; see `<sys/msg.h>`:

45879 **msg_qnum** shall be incremented by 1.

45880 **msg_lspid** shall be set to the process ID of the calling process.

45881 **msg_stime** shall be set to the current time, as described in [Section 2.7.1](#) (on page 502).

45882 RETURN VALUE

45883 Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,
45884 *msgsnd()* shall return -1, and *errno* shall be set to indicate the error.

45885 ERRORS

45886 The *msgsnd()* function shall fail if:

45887 [EACCES] Operation permission is denied to the calling process; see [Section 2.7](#) (on page
45888 501).

45889 [EAGAIN] The message cannot be sent for one of the reasons cited above and (*msgflg* &
45890 IPC_NOWAIT) is non-zero.

45891 [EIDRM] The message queue identifier *msqid* is removed from the system.

45892 [EINTR] The *msgsnd()* function was interrupted by a signal.

45893 [EINVAL] The value of *msqid* is not a valid message queue identifier, or the value of
45894 *mtype* is less than 1; or the value of *msgsz* is greater than the system-imposed
45895 limit.

45896 EXAMPLES

45897 Sending a Message

45898 The following example sends a message to the queue identified by the *msqid* argument
45899 (assuming that value has previously been set). This call specifies that an error should be
45900 reported if no message is available. The message size is calculated directly using the *sizeof*
45901 operator.

```
45902 #include <sys/msg.h>
45903 ...
45904 int result;
45905 int msqid;
45906 struct message {
45907     long type;
45908     char text[20];
45909 } msg;
45910 msg.type = 1;
45911 strcpy(msg.text, "This is message 1");
45912 ...
45913 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

45914 APPLICATION USAGE

45915 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
45916 (IPC). Application developers who need to use IPC should design their applications so that
45917 modules using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to
45918 use the alternative interfaces.

45919 **RATIONALE**

45920 None.

45921 **FUTURE DIRECTIONS**

45922 None.

45923 **SEE ALSO**45924 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
45925 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#),
45926 [sigaction\(\)](#)45927 [XBD Section 3.226](#) (on page 69), [<sys/msg.h>](#)45928 **CHANGE HISTORY**

45929 First released in Issue 2. Derived from Issue 2 of the SVID.

45930 **Issue 5**45931 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
45932 DIRECTIONS to a new APPLICATION USAGE section.45933 **Issue 6**

45934 The normative text is updated to avoid use of the term “must” for application requirements.

45935 **Issue 7**45936 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0400 [345] and XSH/TC1-2008/0401
45937 [359] are applied.

45938 **NAME**45939 `msync` — synchronize memory with physical storage45940 **SYNOPSIS**

```
45941 XSI|SIO #include <sys/mman.h>
45942 int msync(void *addr, size_t len, int flags);
```

45943 **DESCRIPTION**

45944 The `msync()` function shall write all modified data to permanent storage locations, if any, in
 45945 those whole pages containing any part of the address space of the process starting at address
 45946 *addr* and continuing for *len* bytes. If no such storage exists, `msync()` need not have any effect. If
 45947 requested, the `msync()` function shall then invalidate cached copies of data.

45948 The implementation may require that *addr* be a multiple of the page size as returned by
 45949 `sysconf()`.

45950 For mappings to files, the `msync()` function shall ensure that all write operations are completed
 45951 as defined for synchronized I/O data integrity completion. It is unspecified whether the
 45952 implementation also writes out other file attributes. When the `msync()` function is called on
 45953 MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and
 45954 shall not cause such data to be made visible to other processes. It is unspecified whether data in
 45955 SHM|TYM MAP_PRIVATE mappings has any permanent storage locations. The effect of `msync()` on a
 45956 shared memory object or a typed memory object is unspecified. The behavior of this function is
 45957 unspecified if the mapping was not established by a call to `mmap()`.

45958 The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following
 45959 flags defined in the `<sys/mman.h>` header:

45960	Symbolic Constant	Description
45961	MS_ASYNC	Perform asynchronous writes.
45962	MS_SYNC	Perform synchronous writes.
45963	MS_INVALIDATE	Invalidate cached data.

45964 When MS_ASYNC is specified, `msync()` shall return immediately once all the write operations
 45965 are initiated or queued for servicing; when MS_SYNC is specified, `msync()` shall not return until
 45966 all write operations are completed as defined for synchronized I/O data integrity completion.
 45967 Either MS_ASYNC or MS_SYNC shall be specified, but not both.

45968 When MS_INVALIDATE is specified, `msync()` shall invalidate all cached copies of mapped data
 45969 that are inconsistent with the permanent storage locations such that subsequent references shall
 45970 obtain data that was consistent with the permanent storage locations sometime between the call
 45971 to `msync()` and the first subsequent memory reference to the data.

45972 If `msync()` causes any write to a file, the file's last data modification and last file status change
 45973 timestamps shall be marked for update.

45974 **RETURN VALUE**

45975 Upon successful completion, `msync()` shall return 0; otherwise, it shall return `-1` and set *errno* to
 45976 indicate the error.

45977 **ERRORS**

45978 The `msync()` function shall fail if:

45979 [EBUSY] Some or all of the addresses in the range starting at *addr* and continuing for *len*
 45980 bytes are locked, and MS_INVALIDATE is specified.

45981 [EINVAL] The value of *flags* is invalid.
 45982 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are
 45983 outside the range allowed for the address space of a process or specify one or
 45984 more pages that are not mapped.

45985 The *msync()* function may fail if:

45986 [EINVAL] The value of *addr* is not a multiple of the page size as returned by *sysconf()*.

45987 EXAMPLES

45988 None.

45989 APPLICATION USAGE

45990 The *msync()* function is only supported if the Synchronized Input and Output option is
 45991 supported, and thus need not be available on all implementations.

45992 The *msync()* function should be used by programs that require a memory object to be in a
 45993 known state; for example, in building transaction facilities.

45994 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees
 45995 that *msync()* is the only control over when pages are or are not written to disk.

45996 RATIONALE

45997 The *msync()* function writes out data in a mapped region to the permanent storage for the
 45998 underlying object. The call to *msync()* ensures data integrity of the file.

45999 After the data is written out, any cached data may be invalidated if the `MS_INVALIDATE` flag
 46000 was specified. This is useful on systems that do not support read/write consistency.

46001 FUTURE DIRECTIONS

46002 None.

46003 SEE ALSO

46004 [*mmap\(\)*](#), [*sysconf\(\)*](#)

46005 XBD [`<sys/mman.h>`](#)

46006 CHANGE HISTORY

46007 First released in Issue 4, Version 2.

46008 Issue 5

46009 Moved from X/OPEN UNIX extension to BASE.

46010 Aligned with *msync()* in the POSIX Realtime Extension as follows:

46011 The DESCRIPTION is extensively reworded.

46012 [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.

46013 Issue 6

46014 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input
 46015 and Output options.

46016 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

46017 The [EBUSY] mandatory error condition is added.

46018 The following new requirements on POSIX implementations derive from alignment with the
 46019 Single UNIX Specification:

46020 The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
46021 of the page size.

46022 The second [EINVAL] error condition is made mandatory.

46023 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to
46024 typed memory objects.

46025 **Issue 7**

46026 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
46027 requirements.

46028 SD5-XSH-ERN-110 is applied.

46029 The *msync()* function is marked as part of the Synchronized Input and Output option or XSI
46030 option as the Memory Mapped Files is moved to the Base.

46031 Changes are made related to support for finegrained timestamps.

46032 **NAME**

46033 munlock — unlock a range of process address space

46034 **SYNOPSIS**

```
46035 MLR       #include <sys/mman.h>  
46036           int munlock(const void *addr, size_t len);
```

46037 **DESCRIPTION**

46038 Refer to *mlock()*.

46039 **NAME**

46040 munlockall — unlock the address space of a process

46041 **SYNOPSIS**

46042 ML #include <sys/mman.h>

46043 int munlockall(void);

46044 **DESCRIPTION**46045 Refer to *mlockall()*.

46046 **NAME**

46047 munmap †unmap pages of memory

46048 **SYNOPSIS**

46049 #include <sys/mman.h>

46050 int munmap(void *addr, size_t len);

46051 **DESCRIPTION**

46052 The *munmap()* function shall remove any mappings for those entire pages containing any part of
 46053 the address space of the process starting at *addr* and continuing for *len* bytes. Further references
 46054 to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no
 46055 mappings in the specified address range, then *munmap()* has no effect.

46056 The implementation may require that *addr* be a multiple of the page size as returned by
 46057 *sysconf()*.

46058 If a mapping to be removed was private, any modifications made in this address range shall be
 46059 discarded.

46060 MLIMLR Any memory locks (see *mlock()* and *mlockall()*) associated with this address range shall be
 46061 removed, as if by an appropriate call to *munlock()*.

46062 TYM If a mapping removed from a typed memory object causes the corresponding address range of
 46063 the memory pool to be inaccessible by any process in the system except through allocatable
 46064 mappings (that is, mappings of typed memory objects opened with the
 46065 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall
 46066 become deallocated and may become available to satisfy future typed memory allocation
 46067 requests.

46068 A mapping removed from a typed memory object opened with the
 46069 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of
 46070 that typed memory for allocation.

46071 The behavior of this function is unspecified if the mapping was not established by a call to
 46072 *mmap()*.

46073 **RETURN VALUE**

46074 Upon successful completion, *munmap()* shall return 0; otherwise, it shall return -1 and set *errno*
 46075 to indicate the error.

46076 **ERRORS**

46077 The *munmap()* function shall fail if:

46078 [EINVAL] Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the
 46079 address space of a process.

46080 [EINVAL] The *len* argument is 0.

46081 The *munmap()* function may fail if:

46082 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

46083 EXAMPLES

46084 None.

46085 APPLICATION USAGE

46086 None.

46087 RATIONALE

46088 The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

46089 It is possible that an application has applied process memory locking to a region that contains
46090 shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary,
46091 causes those locks to be removed.

46092 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.

46093 FUTURE DIRECTIONS

46094 None.

46095 SEE ALSO

46096 [mlock\(\)](#), [mlockall\(\)](#), [mmap\(\)](#), [posix_typed_mem_open\(\)](#), [sysconf\(\)](#)

46097 XBD [<sys/mman.h>](#)

46098 CHANGE HISTORY

46099 First released in Issue 4, Version 2.

46100 Issue 5

46101 Moved from X/OPEN UNIX extension to BASE.

46102 Aligned with *munmap()* in the POSIX Realtime Extension as follows:

46103 The DESCRIPTION is extensively reworded.

46104 The SIGBUS error is no longer permitted to be generated.

46105 Issue 6

46106 The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory
46107 Objects option.

46108 The following new requirements on POSIX implementations derive from alignment with the
46109 Single UNIX Specification:

46110 The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
46111 of the page size.

46112 The [EINVAL] error conditions are added.

46113 The following changes are made for alignment with IEEE Std 1003.1j-2000:

46114 Semantics for typed memory objects are added to the DESCRIPTION.

46115 The *posix_typed_mem_open()* function is added to the SEE ALSO section.

46116 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code
46117 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).

46118 Issue 7

46119 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
46120 requirements.

46121 The *munmap()* function is moved from the Memory Mapped Files option to the Base.

46122 **NAME**

46123 nan, nanf, nanl — return quiet NaN

46124 **SYNOPSIS**

```
46125 #include <math.h>
46126 double nan(const char *tagp);
46127 float nanf(const char *tagp);
46128 long double nanl(const char *tagp);
```

46129 **DESCRIPTION**

46130 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46131 conflict between the requirements described here and the ISO C standard is unintentional. This
 46132 volume of POSIX.1-2017 defers to the ISO C standard.

46133 The function call `nan("n-char-sequence")` shall be equivalent to:

```
46134 strtod("NAN(n-char-sequence)", (char **) NULL);
```

46135 The function call `nan("")` shall be equivalent to:

```
46136 strtod("NAN()", (char **) NULL)
```

46137 If `tagp` does not point to an *n-char* sequence or an empty string, the function call shall be
 46138 equivalent to:

```
46139 strtod("NAN", (char **) NULL)
```

46140 Function calls to `nanf()` and `nanl()` are equivalent to the corresponding function calls to `strtof()`
 46141 and `strtold()`.

46142 **RETURN VALUE**

46143 These functions shall return a quiet NaN, if available, with content indicated through `tagp`.

46144 If the implementation does not support quiet NaNs, these functions shall return zero.

46145 **ERRORS**

46146 No errors are defined.

46147 **EXAMPLES**

46148 None.

46149 **APPLICATION USAGE**

46150 None.

46151 **RATIONALE**

46152 None.

46153 **FUTURE DIRECTIONS**

46154 None.

46155 **SEE ALSO**

46156 [strtod\(\)](#)

46157 XBD [<math.h>](#)

46158 **CHANGE HISTORY**

46159 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

46160 **NAME**

46161 nanosleep — high resolution sleep

46162 **SYNOPSIS**

```
46163 CX #include <time.h>
46164 int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

46165 **DESCRIPTION**

46166 The *nanosleep()* function shall cause the current thread to be suspended from execution until
 46167 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the
 46168 calling thread, and its action is to invoke a signal-catching function or to terminate the process.
 46169 The suspension time may be longer than requested because the argument value is rounded up to
 46170 an integer multiple of the sleep resolution or because of the scheduling of other activity by the
 46171 system. But, except for the case of being interrupted by a signal, the suspension time shall not be
 46172 less than the time specified by *rqtp*, as measured by the system clock `CLOCK_REALTIME`.

46173 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.

46174 **RETURN VALUE**

46175 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall
 46176 be zero.

46177 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a
 46178 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the
 46179 **timespec** structure referenced by it is updated to contain the amount of time remaining in the
 46180 interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments can
 46181 point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

46182 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.

46183 **ERRORS**

46184 The *nanosleep()* function shall fail if:

- | | | |
|-------|----------|--|
| 46185 | [EINTR] | The <i>nanosleep()</i> function was interrupted by a signal. |
| 46186 | [EINVAL] | The <i>rqtp</i> argument specified a nanosecond value less than zero or greater than
46187 or equal to 1 000 million. |

46188 **EXAMPLES**

46189 None.

46190 **APPLICATION USAGE**

46191 None.

46192 **RATIONALE**

46193 It is common to suspend execution of a thread for an interval in order to poll the status of a non-
 46194 interrupting function. A large number of actual needs can be met with a simple extension to
 46195 *sleep()* that provides finer resolution.

46196 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the
 46197 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,
 46198 it is possible to write such a routine using no static storage and reserving no system facilities.
 46199 Although it is possible to write a function with similar functionality to *sleep()* using the
 46200 remainder of the *timer_**(*)* functions, such a function requires the use of signals and the
 46201 reservation of some signal number. This volume of POSIX.1-2017 requires that *nanosleep()* be
 46202 non-intrusive of the signals function.

46203 The *nanosleep()* function shall return a value of 0 on success and `-1` on failure or if interrupted.

46204 This latter case is different from *sleep()*. This was done because the remaining time is returned
46205 via an argument structure pointer, *rmtpt*, instead of as the return value.

46206 **FUTURE DIRECTIONS**

46207 None.

46208 **SEE ALSO**

46209 *clock_nanosleep()*, *sleep()*

46210 XBD <**time.h**>

46211 **CHANGE HISTORY**

46212 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

46213 **Issue 6**

46214 The *nanosleep()* function is marked as part of the Timers option.

46215 The [ENOSYS] error condition has been removed as stubs need not be provided if an
46216 implementation does not support the Timers option.

46217 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO
46218 section to include the *clock_nanosleep()* function.

46219 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/63 is applied, correcting text in the
46220 RATIONALE section.

46221 **Issue 7**

46222 SD5-XBD-ERN-33 is applied.

46223 The *nanosleep()* function is moved from the Timers option to the Base.

46224 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0231 [909] is applied.

46225 **NAME**

46226 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

46227 **SYNOPSIS**

```
46228 #include <math.h>
46229 double nearbyint(double x);
46230 float nearbyintf(float x);
46231 long double nearbyintl(long double x);
```

46232 **DESCRIPTION**

46233 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46234 conflict between the requirements described here and the ISO C standard is unintentional. This
 46235 volume of POSIX.1-2017 defers to the ISO C standard.

46236 These functions shall round their argument to an integer value in floating-point format, using
 46237 the current rounding direction and without raising the inexact floating-point exception.

46238 **RETURN VALUE**

46239 MX Upon successful completion, these functions shall return the rounded integer value. The result
 46240 shall have the same sign as x .

46241 MX If x is NaN, a NaN shall be returned.

46242 If x is ± 0 , ± 0 shall be returned.

46243 If x is $\pm \text{Inf}$, x shall be returned.

46244 **ERRORS**

46245 No errors are defined.

46246 **EXAMPLES**

46247 None.

46248 **APPLICATION USAGE**

46249 The integral value returned by these functions need not be expressible as an `intmax_t`. The
 46250 return value should be tested before assigning it to an integer type to avoid the undefined
 46251 results of an integer overflow.

46252 **RATIONALE**

46253 None.

46254 **FUTURE DIRECTIONS**

46255 None.

46256 **SEE ALSO**

46257 [feclearexcept\(\)](#), [fetestexcept\(\)](#)

46258 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

46259 **CHANGE HISTORY**

46260 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

46261 **Issue 7**

46262 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0402 [346,428] is applied.

46263 **NAME**

46264 newlocale — create or modify a locale object

46265 **SYNOPSIS**

```
46266 CX #include <locale.h>
46267 locale_t newlocale(int category_mask, const char *locale,
46268 locale_t base);
```

46269 **DESCRIPTION**

46270 The *newlocale()* function shall create a new locale object or modify an existing one. If the *base*
 46271 argument is **(locale_t)0**, a new locale object shall be created. It is unspecified whether the locale
 46272 object pointed to by *base* shall be modified, or freed and a new locale object created.

46273 The *category_mask* argument specifies the locale categories to be set or modified. Values for
 46274 *category_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants
 46275 *LC_CTYPE_MASK*, *LC_NUMERIC_MASK*, *LC_TIME_MASK*, *LC_COLLATE_MASK*,
 46276 *LC_MONETARY_MASK*, and *LC_MESSAGES_MASK*, or any of the implementation-defined
 46277 mask values defined in **<locale.h>**.

46278 For each category with the corresponding bit set in *category_mask* the data from the locale named
 46279 by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale
 46280 named by *locale* shall replace the existing data within the locale object. If a completely new locale
 46281 object is created, the data for all sections not requested by *category_mask* shall be taken from the
 46282 default locale.

46283 The following preset values of *locale* are defined for all settings of *category_mask*:

46284 "POSIX" Specifies the minimal environment for C-language translation called the
 46285 POSIX locale.

46286 "C" Equivalent to "POSIX".

46287 "" Specifies an implementation-defined native environment. This corresponds to
 46288 the value of the associated environment variables, *LC_** and *LANG*; see XBD
 46289 [Chapter 7](#) (on page 135) and [Chapter 8](#) (on page 173).

46290 If the *base* argument is not **(locale_t)0** and the *newlocale()* function call succeeds, the contents of
 46291 *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before
 46292 calling *newlocale()*. If the function call fails and the *base* argument is not **(locale_t)0**, the contents
 46293 of *base* shall remain valid and unchanged.

46294 The behavior is undefined if the *base* argument is the special locale object
 46295 *LC_GLOBAL_LOCALE*, or is not a valid locale object handle and is not **(locale_t)0**.

46296 **RETURN VALUE**

46297 Upon successful completion, the *newlocale()* function shall return a handle which the caller may
 46298 use on subsequent calls to *duplocale()*, *freelocale()*, and other functions taking a **locale_t**
 46299 argument.

46300 Upon failure, the *newlocale()* function shall return **(locale_t)0** and set *errno* to indicate the error.

46301 **ERRORS**

46302 The *newlocale()* function shall fail if:

46303 [ENOMEM] There is not enough memory available to create the locale object or load the
 46304 locale data.

46305 [EINVAL] The *category_mask* contains a bit that does not correspond to a valid category.
 46306 [ENOENT] For any of the categories in *category_mask*, the locale data is not available.
 46307 The *newlocale()* function may fail if:
 46308 [EINVAL] The *locale* argument is not a valid string pointer.

46309 EXAMPLES

46310 Constructing a Locale Object from Different Locales

46311 The following example shows the construction of a locale where the *LC_CTYPE* category data
 46312 comes from a locale *loc1* and the *LC_TIME* category data from a locale *loc2*:

```
46313 #include <locale.h>
46314 ...
46315 locale_t loc, new_loc;
46316 /* Get the "loc1" data. */
46317 loc = newlocale (LC_CTYPE_MASK, "loc1", (locale_t)0);
46318 if (loc == (locale_t) 0)
46319     abort ();
46320 /* Get the "loc2" data. */
46321 new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
46322 if (new_loc != (locale_t) 0)
46323     /* We don t abort if this fails. In this case this
46324     simply used to unchanged locale object. */
46325     loc = new_loc;
46326 ...
```

46327 Freeing up a Locale Object

46328 The following example shows a code fragment to free a locale object created by *newlocale()*:

```
46329 #include <locale.h>
46330 ...
46331 /* Every locale object allocated with newlocale() should be
46332 * freed using freelocale():
46333 */
46334 locale_t loc;
46335 /* Get the locale. */
46336 loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", (locale_t)0);
46337 /* ... Use the locale object ... */
46338 ...
46339 /* Free the locale object resources. */
46340 freelocale (loc);
```


46341 **APPLICATION USAGE**

46342 Handles for locale objects created by the *newlocale()* function should either be released by a
46343 corresponding call to *freelocale()*, or be used as a base locale to another *newlocale()* call.

46344 The special locale object LC_GLOBAL_LOCALE must not be passed for the *base* argument, even
46345 when returned by the *uselocale()* function.

46346 **RATIONALE**

46347 None.

46348 **FUTURE DIRECTIONS**

46349 None.

46350 **SEE ALSO**

46351 *duplocale()*, *freelocale()*, *uselocale()*

46352 XBD [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [<locale.h>](#)

46353 **CHANGE HISTORY**

46354 First released in Issue 7.

46355 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0403 [227], XSH/TC1-2008/0404 [283],
46356 XSH/TC1-2008/0405 [295], and XSH/TC1-2008/0406 [227] are applied.

46357 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0232 [781] and XSH/TC2-2008/0233
46358 [673] are applied.

46359 **NAME**

46360 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable
 46361 floating-point number

46362 **SYNOPSIS**

```
46363 #include <math.h>
46364 double nextafter(double x, double y);
46365 float nextafterf(float x, float y);
46366 long double nextafterl(long double x, long double y);
46367 double nexttoward(double x, long double y);
46368 float nexttowardf(float x, long double y);
46369 long double nexttowardl(long double x, long double y);
```

46370 **DESCRIPTION**

46371 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 46372 conflict between the requirements described here and the ISO C standard is unintentional. This
 46373 volume of POSIX.1-2017 defers to the ISO C standard.

46374 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable
 46375 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall
 46376 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,
 46377 and *nextafterl()* functions shall return *y* if *x* equals *y*.

46378 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the
 46379 corresponding *nextafter()* functions, except that the second parameter shall have type **long**
 46380 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

46381 An application wishing to check for error situations should set *errno* to zero and call
 46382 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 46383 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 46384 zero, an error has occurred.

46385 **RETURN VALUE**

46386 Upon successful completion, these functions shall return the next representable floating-point
 46387 value following *x* in the direction of *y*.

46388 If $x=y$, *y* (of the type *x*) shall be returned.

46389 If *x* is finite and the correct function value would overflow, a range error shall occur and
 46390 \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL (with the same sign as *x*) shall be returned as
 46391 appropriate for the return type of the function.

46392 MX If *x* or *y* is NaN, a NaN shall be returned.

46393 MX If $x \neq y$ and the correct function value is subnormal, zero, or underflows, a range error shall
 46394 occur, and

46395 MXX the correct function value (if representable) or

46396 MX 0.0 shall be returned.

46397 **ERRORS**

46398 These functions shall fail if:

46399 Range Error The correct value overflows.

46400 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 46401 then *errno* shall be set to [ERANGE]. If the integer expression
 46402 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 46403 floating-point exception shall be raised.

46404 MX **Range Error** The correct value is subnormal or underflows.
 46405 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 46406 then *errno* shall be set to [ERANGE]. If the integer expression
 46407 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 46408 floating-point exception shall be raised.

EXAMPLES

46410 None.

APPLICATION USAGE

46412 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 46413 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

46414 When `<tgmath.h>` is included, note that the return type of *nextafter()* depends on the generic
 46415 typing deduced from both arguments, while the return type of *nexttoward()* depends only on the
 46416 generic typing of the first argument.

RATIONALE

46418 None.

FUTURE DIRECTIONS

46420 None.

SEE ALSO

46422 [*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#)

46423 XBD [Section 4.20](#) (on page 117), `<math.h>`, `<tgmath.h>`

CHANGE HISTORY

46425 First released in Issue 4, Version 2.

Issue 5

46427 Moved from X/OPEN UNIX extension to BASE.

Issue 6

46429 The *nextafter()* function is no longer marked as an extension.

46430 The *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions are added
 46431 for alignment with the ISO/IEC 9899:1999 standard.

46432 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 46433 revised to align with the ISO/IEC 9899:1999 standard.

46434 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 46435 marked.

Issue 7

46437 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0407 [68] and XSH/TC1-2008/0408
 46438 [357] are applied.

46439 **NAME**

46440 nftw — walk a file tree

46441 **SYNOPSIS**

```
46442 XSI      #include <ftw.h>
46443
46443      int nftw(const char *path, int (*fn)(const char *,
46444              const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

46445 **DESCRIPTION**

46446 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*
 46447 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a
 46448 bitwise-inclusive OR of zero or more of the following flags:

46449 **FTW_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it
 46450 reports files in that directory. If clear, *nftw()* shall not change the current
 46451 working directory.

46452 **FTW_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory
 46453 itself. If clear, *nftw()* shall report any directory before reporting the files in that
 46454 directory.

46455 **FTW_MOUNT** If set, *nftw()* shall only report files in the same file system as *path*. If clear,
 46456 *nftw()* shall report all files encountered during the walk.

46457 **FTW_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

46458 If **FTW_PHYS** is clear and **FTW_DEPTH** is set, *nftw()* shall follow links instead of reporting
 46459 them, but shall not report any directory that would be a descendant of itself. If **FTW_PHYS** is
 46460 clear and **FTW_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not
 46461 report the contents of any directory that would be a descendant of itself.

46462 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

46463 The first argument is the pathname of the object.

46464 The second argument is a pointer to the **stat** buffer containing information on the object,
 46465 filled in as if *fstatat()*, *stat()*, or *lstat()* had been called to retrieve the information.

46466 The third argument is an integer giving additional information. Its value is one of the
 46467 following:

46468 **FTW_D** The object is a directory.

46469 **FTW_DNR** The object is a directory that cannot be read. The *fn* function shall not be
 46470 called for any of its descendants.

46471 **FTW_DP** The object is a directory and subdirectories have been visited. (This condition
 46472 shall only occur if the **FTW_DEPTH** flag is included in *flags*.)

46473 **FTW_F** The object is a non-directory file.

46474 **FTW_NS** The *stat()* function failed on the object because of lack of appropriate
 46475 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any
 46476 other reason is considered an error and *nftw()* shall return -1 .

46477 **FTW_SL** The object is a symbolic link. (This condition shall only occur if the
 46478 **FTW_PHYS** flag is included in *flags*.)

46479 FTW_SLN The object is a symbolic link that does not name an existing file. (This
46480 condition shall only occur if the FTW_PHYS flag is not included in *flags*.)

46481 The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the
46482 object's filename in the pathname passed as the first argument to *fn*. The value of **level**
46483 indicates depth relative to the root of the walk, where the root level is 0.

46484 The results are unspecified if the application-supplied *fn* function does not preserve the current
46485 working directory.

46486 The argument *fd_limit* sets the maximum number of file descriptors that shall be used by *nftw()*
46487 while traversing the file tree. At most one file descriptor shall be used for each directory level.

46488 The *nftw()* function need not be thread-safe.

46489 RETURN VALUE

46490 The *nftw()* function shall continue until the first of the following conditions occurs:

46491 An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that
46492 value.

46493 The *nftw()* function detects an error other than [EACCES] (see FTW_DNR and FTW_NS
46494 above), in which case *nftw()* shall return -1 and set *errno* to indicate the error.

46495 The tree is exhausted, in which case *nftw()* shall return 0.

46496 ERRORS

46497 The *nftw()* function shall fail if:

46498 [EACCES] Search permission is denied for any component of *path* or read permission is
46499 denied for *path*, or *fn* returns -1 and does not reset *errno*.

46500 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
46501 argument.

46502 [ENAMETOOLONG]
46503 The length of a component of a pathname is longer than {NAME_MAX}.

46504 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

46505 [ENOTDIR] A component of *path* names an existing file that is neither a directory nor a
46506 symbolic link to a directory.

46507 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
46508 programming environment for one or more files found in the file hierarchy.

46509 The *nftw()* function may fail if:

46510 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
46511 resolution of the *path* argument.

46512 [EMFILE] All file descriptors available to the process are currently open.

46513 [ENAMETOOLONG]
46514 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
46515 symbolic link produced an intermediate result with a length that exceeds
46516 {PATH_MAX}.

46517 [ENFILE] Too many files are currently open in the system.

46518 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

46519 **EXAMPLES**

46520 The following program traverses the directory tree under the path named in its first command-
 46521 line argument, or under the current directory if no argument is supplied. It displays various
 46522 information about each file. The second command-line argument can be used to specify
 46523 characters that control the value assigned to the *flags* argument when calling *nftw()*.

```

46524 #include <ftw.h>
46525 #include <stdio.h>
46526 #include <stdlib.h>
46527 #include <string.h>
46528 #include <stdint.h>

46529 static int
46530 display_info(const char *fpath, const struct stat *sb,
46531             int tflag, struct FTW *ftwbuf)
46532 {
46533     printf("%-3s %2d %7jd  %-40s %d %s\n",
46534           (tflag == FTW_D) ? "d" : (tflag == FTW_DNR) ? "dnr" :
46535           (tflag == FTW_DP) ? "dp" : (tflag == FTW_F) ?
46536           (S_ISBLK(sb->st_mode) ? "f b" :
46537           S_ISCHR(sb->st_mode) ? "f c" :
46538           S_ISFIFO(sb->st_mode) ? "f p" :
46539           S_ISREG(sb->st_mode) ? "f r" :
46540           S_ISSOCK(sb->st_mode) ? "f s" : "f ?") :
46541           (tflag == FTW_NS) ? "ns" : (tflag == FTW_SL) ? "sl" :
46542           (tflag == FTW_SLN) ? "sln" : "?",
46543           ftwbuf->level, (intmax_t) sb->st_size,
46544           fpath, ftwbuf->base, fpath + ftwbuf->base);
46545     return 0; /* To tell nftw() to continue */
46546 }

46547 int
46548 main(int argc, char *argv[])
46549 {
46550     int flags = 0;

46551     if (argc > 2 && strchr(argv[2], 'd') != NULL)
46552         flags |= FTW_DEPTH;
46553     if (argc > 2 && strchr(argv[2], 'p') != NULL)
46554         flags |= FTW_PHYS;

46555     if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
46556     {
46557         perror("nftw");
46558         exit(EXIT_FAILURE);
46559     }

46560     exit(EXIT_SUCCESS);
46561 }

```

46562 **APPLICATION USAGE**

46563 The *nftw()* function may allocate dynamic storage during its operation. If *nftw()* is forcibly
 46564 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*
 46565 or an interrupt routine, *nftw()* does not have a chance to free that storage, so it remains
 46566 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has

46567 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next
46568 invocation.

46569 **RATIONALE**

46570 None.

46571 **FUTURE DIRECTIONS**

46572 None.

46573 **SEE ALSO**

46574 *fdopendir()*, *fstatat()*, *readdir()*

46575 XBD <[ftw.h](#)>

46576 **CHANGE HISTORY**

46577 First released in Issue 4, Version 2.

46578 **Issue 5**

46579 Moved from X/OPEN UNIX extension to BASE.

46580 In the DESCRIPTION, the definition of the *depth* argument is clarified.

46581 **Issue 6**

46582 The Open Group Base Resolution bwg97-003 is applied.

46583 The ERRORS section is updated as follows:

46584 The wording of the mandatory [ELOOP] error condition is updated.

46585 A second optional [ELOOP] error condition is added.

46586 The [EOVERFLOW] mandatory error condition is added.

46587 Text is added to the DESCRIPTION to say that the *nftw()* function need not be reentrant and that
46588 the results are unspecified if the application-supplied *fn* function does not preserve the current
46589 working directory.

46590 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/64 is applied, changing the argument
46591 *depth* to *fd_limit* throughout and changing “to a maximum of 5 levels deep” to “using a
46592 maximum of 5 file descriptors” in the EXAMPLES section.

46593 **Issue 7**

46594 Austin Group Interpretations 1003.1-2001 #143 and #156 are applied.

46595 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

46596 SD5-XBD-ERN-61 is applied.

46597 APPLICATION USAGE is added and the EXAMPLES section is replaced with a new example.

46598 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0409 [403], XSH/TC1-2008/0410 [324],
46599 and XSH/TC1-2008/0411 [403] are applied.

46600 **NAME**

46601 nice — change the nice value of a process

46602 **SYNOPSIS**

```
46603 XSI      #include <unistd.h>
46604         int nice(int incr);
```

46605 **DESCRIPTION**

46606 The *nice()* function shall add the value of *incr* to the nice value of the calling process. A nice
 46607 value of a process is a non-negative number for which a more positive value shall result in less
 46608 favorable scheduling.

46609 A maximum nice value of $2^{\{NZERO\}}-1$ and a minimum nice value of 0 shall be imposed by the
 46610 system. Requests for values above or below these limits shall result in the nice value being set to
 46611 the corresponding limit. Only a process with appropriate privileges can lower the nice value.

46612 PS|TPS Calling the *nice()* function has no effect on the priority of processes or threads with policy
 46613 SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies
 46614 is implementation-defined.

46615 The nice value set with *nice()* shall be applied to the process. If the process is multi-threaded, the
 46616 nice value shall affect all system scope threads in the process.

46617 As -1 is a permissible return value in a successful situation, an application wishing to check for
 46618 error situations should set *errno* to 0, then call *nice()*, and if it returns -1 , check to see whether
 46619 *errno* is non-zero.

46620 **RETURN VALUE**

46621 Upon successful completion, *nice()* shall return the new nice value $-\{NZERO\}$. Otherwise, -1
 46622 shall be returned, the nice value of the process shall not be changed, and *errno* shall be set to
 46623 indicate the error.

46624 **ERRORS**

46625 The *nice()* function shall fail if:

46626 [EPERM] The *incr* argument is negative and the calling process does not have
 46627 appropriate privileges.

46628 **EXAMPLES**46629 **Changing the Nice Value**

46630 The following example adds the value of the *incr* argument, -20 , to the nice value of the calling
 46631 process.

```
46632 #include <unistd.h>
46633 ...
46634 int incr = -20;
46635 int ret;
46636
46637 ret = nice(incr);
```

46637 **APPLICATION USAGE**

46638 None.

46639 **RATIONALE**

46640 None.

46641 **FUTURE DIRECTIONS**

46642 None.

46643 **SEE ALSO**46644 *exec*, *getpriority()*46645 XBD [<limits.h>](#), [<unistd.h>](#)46646 **CHANGE HISTORY**

46647 First released in Issue 1. Derived from Issue 1 of the SVID.

46648 **Issue 5**46649 A statement is added to the description indicating the effects of this function on the different
46650 scheduling policies and multi-threaded processes.

46651 **NAME**

46652 nl_langinfo, nl_langinfo_l †'language information

46653 **SYNOPSIS**

46654 #include <langinfo.h>

46655 char *nl_langinfo(nl_item item);

46656 char *nl_langinfo_l(nl_item item, locale_t locale);

46657 **DESCRIPTION**

46658 The *nl_langinfo()* and *nl_langinfo_l()* functions shall return a pointer to a string containing
 46659 information relevant to the particular language or cultural area defined in the current locale, or
 46660 in the locale represented by *locale*, respectively (see <langinfo.h>). The manifest constant names
 46661 and values of *item* are defined in <langinfo.h>. For example:

46662 nl_langinfo(ABDAY_1)

46663 would return a pointer to the string "Dom" if the identified language was Portuguese, and
 46664 "Sun" if the identified language was English.

46665 nl_langinfo_l(ABDAY_1, loc)

46666 would return a pointer to the string "Dom" if the identified language of the locale represented by
 46667 *loc* was Portuguese, and "Sun" if the identified language of the locale represented by *loc* was
 46668 English.

46669 The *nl_langinfo()* function need not be thread-safe.

46670 The behavior is undefined if the *locale* argument to *nl_langinfo_l()* is the special locale object
 46671 LC_GLOBAL_LOCALE or is not a valid locale object handle.

46672 **RETURN VALUE**

46673 In a locale where *langinfo* data is not defined, these functions shall return a pointer to the
 46674 corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to
 46675 an empty string if *item* contains an invalid setting.

46676 The application shall not modify the string returned. The pointer returned by *nl_langinfo()*
 46677 might be invalidated or the string content might be overwritten by a subsequent call to
 46678 *nl_langinfo()* in any thread or to *nl_langinfo_l()* in the same thread or the initial thread, by
 46679 subsequent calls to *setlocale()* with a category corresponding to the category of *item* (see
 46680 <langinfo.h>) or the category LC_ALL, or by subsequent calls to *uselocale()* which change the
 46681 category corresponding to the category of *item*. The pointer returned by *nl_langinfo_l()* might be
 46682 invalidated or the string content might be overwritten by a subsequent call to *nl_langinfo_l()* in
 46683 the same thread or to *nl_langinfo()* in any thread, or by subsequent calls to *freelocale()* or
 46684 *newlocale()* which free or modify the locale object that was passed to *nl_langinfo_l()*. The
 46685 returned pointer and the string content might also be invalidated if the calling thread is
 46686 terminated.

46687 **ERRORS**

46688 No errors are defined.

46689 **EXAMPLES**46690 **Getting Date and Time Formatting Information**

46691 The following example returns a pointer to a string containing date and time formatting
46692 information, as defined in the *LC_TIME* category of the current locale.

```
46693 #include <time.h>
46694 #include <langinfo.h>
46695 ...
46696 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
46697 ...
```

46698 **APPLICATION USAGE**

46699 The array pointed to by the return value should not be modified by the program, but may be
46700 modified by further calls to these functions.

46701 **RATIONALE**

46702 The possible interactions between internal data used by *nl_langinfo()* and *nl_langinfo_l()* are
46703 complicated by the fact that *nl_langinfo_l()* must be thread-safe but *nl_langinfo()* need not be.
46704 The various implementation choices are:

- 46705 1. *nl_langinfo_l()* and *nl_langinfo()* use separate buffers, or at least one of them does not use
46706 an internal string buffer. In this case there are no interactions.
- 46707 2. *nl_langinfo_l()* and *nl_langinfo()* share an internal per-thread buffer. There can be
46708 interactions, but only in the same thread.
- 46709 3. *nl_langinfo_l()* uses an internal per-thread buffer, and *nl_langinfo()* uses (in all threads)
46710 the same buffer that *nl_langinfo_l()* uses in the initial thread. There can be interactions,
46711 but only when *nl_langinfo_l()* is called in the initial thread.

46712 **FUTURE DIRECTIONS**

46713 None.

46714 **SEE ALSO**

46715 [*setlocale\(\)*](#), [*uselocale\(\)*](#)

46716 XBD [Chapter 7](#) (on page 135), [<langinfo.h>](#), [<locale.h>](#), [<nl_types.h>](#)

46717 **CHANGE HISTORY**

46718 First released in Issue 2.

46719 **Issue 5**

46720 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

46721 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

46722 **Issue 7**

46723 Austin Group Interpretation 1003.1-2001 #156 is applied.

46724 The *nl_langinfo()* function is moved from the XSI option to the Base.

46725 The *nl_langinfo_l()* function is added from The Open Group Technical Standard, 2006, Extended
46726 API Set Part 4.

46727 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0412 [302], XSH/TC1-2008/0413 [75],
46728 XSH/TC1-2008/0414 [283], XSH/TC1-2008/0415 [75,402], XSH/TC1-2008/0416 [283], and
46729 XSH/TC1-2008/0417 [402] are applied.

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0234 [656] is applied.

46731 **NAME**

46732 nrand48 ‡generate uniformly distributed pseudo-random non-negative long integers

46733 **SYNOPSIS**

```
46734 XSI #include <stdlib.h>  
46735 long nrand48(unsigned short xsubi[3]);
```

46736 **DESCRIPTION**

46737 Refer to *drand48()*.

46738 **NAME**

46739 ntohl, ntohs — convert values between host and network byte order

46740 **SYNOPSIS**

46741 #include <arpa/inet.h>

46742 uint32_t ntohl(uint32_t *netlong*);46743 uint16_t ntohs(uint16_t *netshort*);46744 **DESCRIPTION**46745 Refer to *htonl()*.

46746 **NAME**

46747 open, openat ‡'open file

46748 **SYNOPSIS**

46749 OH #include <sys/stat.h>

46750 #include <fcntl.h>

46751 int open(const char *path, int oflag, ...);

46752 int openat(int fd, const char *path, int oflag, ...);

46753 **DESCRIPTION**

46754 The *open()* function shall establish the connection between a file and a file descriptor. It shall
 46755 create an open file description that refers to a file and a file descriptor that refers to that open file
 46756 description. The file descriptor is used by other I/O functions to refer to that file. The *path*
 46757 argument points to a pathname naming the file.

46758 The *open()* function shall return a file descriptor for the named file, allocated as described in
 46759 [Section 2.14](#) (on page 549). The open file description is new, and therefore the file descriptor
 46760 shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag
 46761 associated with the new file descriptor shall be cleared unless the O_CLOEXEC flag is set in
 46762 *oflag*.

46763 The file offset used to mark the current position within the file shall be set to the beginning of
 46764 the file.

46765 The file status flags and file access modes of the open file description shall be set according to
 46766 the value of *oflag*.

46767 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 46768 in <fcntl.h>. Applications shall specify exactly one of the first five values (file access modes)
 46769 below in the value of *oflag*:

46770 O_EXEC Open for execute only (non-directory files). The result is unspecified if
 46771 this flag is applied to a directory.

46772 O_RDONLY Open for reading only.

46773 O_RDWR Open for reading and writing. The result is undefined if this flag is
 46774 applied to a FIFO.

46775 O_SEARCH Open directory for search only. The result is unspecified if this flag is
 46776 applied to a non-directory file.

46777 O_WRONLY Open for writing only.

46778 Any combination of the following may be used:

46779 O_APPEND If set, the file offset shall be set to the end of the file prior to each write.

46780 O_CLOEXEC If set, the FD_CLOEXEC flag for the new file descriptor shall be set.

46781 O_CREAT If the file exists, this flag has no effect except as noted under O_EXCL
 46782 below. Otherwise, if O_DIRECTORY is not set the file shall be created as a
 46783 regular file; the user ID of the file shall be set to the effective user ID of the
 46784 process; the group ID of the file shall be set to the group ID of the file's
 46785 parent directory or to the effective group ID of the process; and the access
 46786 permission bits (see <sys/stat.h>) of the file mode shall be set to the value
 46787 of the argument following the *oflag* argument taken as type **mode_t**
 46788 modified as follows: a bitwise AND is performed on the file-mode bits
 46789 and the corresponding bits in the complement of the process' file mode

46790			creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. When bits other than the file permission bits are set, the effect is unspecified. The argument following the <i>oflag</i> argument does not affect whether the file is open for reading, writing, or for both. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process.
46791			
46792			
46793			
46794			
46795			
46796			
46797			
46798			
46799		O_DIRECTORY	If <i>path</i> resolves to a non-directory file, fail and set <i>errno</i> to [ENOTDIR].
46800	SIO	O_DSYNC	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.
46801			
46802		O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing <i>open()</i> naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.
46803			
46804			
46805			
46806			
46807			
46808			
46809			
46810		O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the terminal device to become the controlling terminal for the process. If <i>path</i> does not identify a terminal device, O_NOCTTY shall be ignored.
46811			
46812			
46813		O_NOFOLLOW	If <i>path</i> names a symbolic link, fail and set <i>errno</i> to [ELOOP].
46814		O_NONBLOCK	When opening a FIFO with O_RDONLY or O_WRONLY set: If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return without delay. An <i>open()</i> for writing-only shall return an error if no process currently has the file open for reading. If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the calling thread until a thread opens the file for writing. An <i>open()</i> for writing-only shall block the calling thread until a thread opens the file for reading.
46815			
46816			
46817			
46818			
46819			
46820			
46821			
46822			When opening a block special or character special file that supports non-blocking opens:
46823			
46824			If O_NONBLOCK is set, the <i>open()</i> function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.
46825			
46826			
46827			If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling thread until the device is ready or available before returning.
46828			
46829			Otherwise, the O_NONBLOCK flag shall not cause an error, but it is unspecified whether the file status flags will include the O_NONBLOCK flag.
46830			
46831			
46832	SIO	O_RSYNC	Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file
46833			
46834			

46835			descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
46836			
46837			
46838			
46839	XSI SIO	O_SYNC	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
46840			
46841	XSI		The O_SYNC flag shall be supported for regular files, even if the Synchronized Input and Output option is not supported.
46842			
46843		O_TRUNC	If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC without either O_RDWR or O_WRONLY is undefined.
46844			
46845			
46846			
46847			
46848			
46849		O_TTY_INIT	If <i>path</i> identifies a terminal device other than a pseudo-terminal, the device is not already open in any process, and either O_TTY_INIT is set in <i>oflag</i> or O_TTY_INIT has the value zero, <i>open()</i> shall set any non-standard termios structure terminal parameters to a state that provides conforming behavior; see XBD Section 11.2 (on page 205). It is unspecified whether O_TTY_INIT has any effect if the device is already open in any process. If <i>path</i> identifies the slave side of a pseudo-terminal that is not already open in any process, <i>open()</i> shall set any non-standard termios structure terminal parameters to a state that provides conforming behavior, regardless of whether O_TTY_INIT is set. If <i>path</i> does not identify a terminal device, O_TTY_INIT shall be ignored.
46850			
46851			
46852			
46853			
46854			
46855			
46856			
46857			
46858			
46859			
46860			If O_CREAT and O_DIRECTORY are set and the requested access mode is neither O_WRONLY nor O_RDWR, the result is unspecified.
46861			
46862			If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last data modification and last file status change timestamps of the parent directory.
46863			
46864			
46865			
46866			If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall mark for update the last data modification and last file status change timestamps of the file.
46867			
46868	SIO		If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.
46869	OB XSR		If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.
46870			
46871			
46872			
46873			
46874			The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal device since system boot or since the device was closed by the process that last had it open. The application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If <i>path</i> names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i> locks the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i> before opening the slave side.
46875			
46876	XSI		
46877			
46878			
46879			

46880 The largest value that can be represented correctly in an object of type `off_t` shall be established
46881 as the offset maximum in the open file description.

46882 The `openat()` function shall be equivalent to the `open()` function except in the case where `path`
46883 specifies a relative path. In this case the file to be opened is determined relative to the directory
46884 associated with the file descriptor `fd` instead of the current working directory. If the access mode
46885 of the open file description associated with the file descriptor is not `O_SEARCH`, the function
46886 shall check whether directory searches are permitted using the current permissions of the
46887 directory underlying the file descriptor. If the access mode is `O_SEARCH`, the function shall not
46888 perform the check.

46889 The `oflag` parameter and the optional fourth parameter correspond exactly to the parameters of
46890 `open()`.

46891 If `openat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working
46892 directory shall be used and the behavior shall be identical to a call to `open()`.

46893 RETURN VALUE

46894 Upon successful completion, these functions shall open the file and return a non-negative
46895 integer representing the file descriptor. Otherwise, these functions shall return `-1` and set `errno`
46896 to indicate the error. If `-1` is returned, no files shall be created or modified.

46897 ERRORS

46898 These functions shall fail if:

46899 [EACCES] Search permission is denied on a component of the path prefix, or the file
46900 exists and the permissions specified by `oflag` are denied, or the file does not
46901 exist and write permission is denied for the parent directory of the file to be
46902 created, or `O_TRUNC` is specified and write permission is denied.

46903 [EEXIST] `O_CREAT` and `O_EXCL` are set, and the named file exists.

46904 [EINTR] A signal was caught during `open()`.

46905 SIO [EINVAL] The implementation does not support synchronized I/O for this file.

46906 OB XSR [EIO] The `path` argument names a STREAMS file and a hangup or error occurred
46907 during the `open()`.

46908 [EISDIR] The named file is a directory and `oflag` includes `O_WRONLY` or `O_RDWR`, or
46909 includes `O_CREAT` without `O_DIRECTORY`.

46910 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
46911 argument, or `O_NOFOLLOW` was specified and the `path` argument names a
46912 symbolic link.

46913 [EMFILE] All file descriptors available to the process are currently open.

46914 [ENAMETOOLONG]

46915 The length of a component of a pathname is longer than `{NAME_MAX}`.

46916 [ENFILE] The maximum allowable number of files is currently open in the system.

46917 [ENOENT] `O_CREAT` is not set and a component of `path` does not name an existing file, or
46918 `O_CREAT` is set and a component of the path prefix of `path` does not name an
46919 existing file, or `path` points to an empty string.

46920 [ENOENT] or [ENOTDIR]

46921 `O_CREAT` is set, and the `path` argument contains at least one non-`<slash>`
46922 character and ends with one or more trailing `<slash>` characters. If `path`

46923			without the trailing <slash> characters would name an existing file, an
46924			[ENOENT] error shall not occur.
46925	OB XSR	[ENOSR]	The <i>path</i> argument names a STREAMS-based file and the system is unable to
46926			allocate a STREAM.
46927		[ENOSPC]	The directory or file system that would contain the new file cannot be
46928			expanded, the file does not exist, and O_CREAT is specified.
46929		[ENOTDIR]	A component of the path prefix names an existing file that is neither a
46930			directory nor a symbolic link to a directory; or O_CREAT and O_EXCL are not
46931			specified, the <i>path</i> argument contains at least one non-<slash> character and
46932			ends with one or more trailing <slash> characters, and the last pathname
46933			component names an existing file that is neither a directory nor a symbolic
46934			link to a directory; or O_DIRECTORY was specified and the <i>path</i> argument
46935			resolves to a non-directory file.
46936		[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no
46937			process has the file open for reading.
46938		[ENXIO]	The named file is a character special or block special file, and the device
46939			associated with this special file does not exist.
46940		[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented
46941			correctly in an object of type off_t .
46942		[EROFS]	The named file resides on a read-only file system and either O_WRONLY,
46943			O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i>
46944			argument.
46945			The <i>openat()</i> function shall fail if:
46946		[EACCES]	The access mode of the open file description associated with <i>fd</i> is not
46947			O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit
46948			directory searches.
46949		[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is
46950			neither AT_FDCWD nor a valid file descriptor open for reading or searching.
46951		[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated
46952			with a non-directory file.
46953			These functions may fail if:
46954	XSI	[EAGAIN]	The <i>path</i> argument names the slave side of a pseudo-terminal device that is
46955			locked.
46956		[EINVAL]	The value of the <i>oflag</i> argument is not valid.
46957		[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
46958			resolution of the <i>path</i> argument.
46959		[ENAMETOOLONG]	
46960			The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
46961			symbolic link produced an intermediate result with a length that exceeds
46962			{PATH_MAX}.
46963	OB XSR	[ENOMEM]	The <i>path</i> argument names a STREAMS file and the system is unable to allocate
46964			resources.

46965 [EOPNOTSUPP] The *path* argument names a socket.
 46966 [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *oflag* is
 46967 O_WRONLY or O_RDWR.

46968 EXAMPLES

46969 Opening a File for Writing by the Owner

46970 The following example opens the file **/tmp/file**, either by creating it (if it does not already exist),
 46971 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,
 46972 the access permission bits in the file mode of the file are set to permit reading and writing by the
 46973 owner, and to permit reading only by group members and others.

46974 If the call to *open()* is successful, the file is opened for writing.

```
46975 #include <fcntl.h>
46976 ...
46977 int fd;
46978 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
46979 char *pathname = "/tmp/file";
46980 ...
46981 fd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
46982 ...
```

46983 Opening a File Using an Existence Check

46984 The following example uses the *open()* function to try to create the **LOCKFILE** file and open it
 46985 for writing. Since the *open()* function specifies the O_EXCL flag, the call fails if the file already
 46986 exists. In that case, the program assumes that someone else is updating the password file and
 46987 exits.

```
46988 #include <fcntl.h>
46989 #include <stdio.h>
46990 #include <stdlib.h>
46991 #define LOCKFILE "/etc/ptmp"
46992 ...
46993 int pfd; /* Integer for file descriptor returned by open() call. */
46994 ...
46995 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
46996     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
46997 {
46998     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
46999     exit(1);
47000 }
47001 ...
```

47002 **Opening a File for Writing**

47003 The following example opens a file for writing, creating the file if it does not already exist. If the
47004 file does exist, the system truncates the file to zero bytes.

```
47005 #include <fcntl.h>
47006 #include <stdio.h>
47007 #include <stdlib.h>
47008 #define LOCKFILE "/etc/ptmp"
47009 ...
47010 int pfd;
47011 char pathname[PATH_MAX+1];
47012 ...
47013 if ((pfd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC,
47014               S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
47015 {
47016     perror("Cannot open output file\n"); exit(1);
47017 }
47018 ...
```

47019 **APPLICATION USAGE**

47020 POSIX.1-2017 does not require that terminal parameters be automatically set to any state on first
47021 open, nor that they be reset after the last close. It is possible for a non-conforming application to
47022 leave a terminal device in a state where the next process to use that device finds it in a non-
47023 conforming state, but has no way of determining this. To ensure that the device is set to a
47024 conforming initial state, applications which perform a first open of a terminal (other than a
47025 pseudo-terminal) should do so using the O_TTY_INIT flag to set the parameters associated with
47026 the terminal to a conforming state.

47027 Except as specified in this volume of POSIX.1-2017, the flags allowed in *oflag* are not mutually-
47028 exclusive and any number of them may be used simultaneously. Not all combinations of flags
47029 make sense. For example, using O_SEARCH | O_CREAT will successfully open a pre-existing
47030 directory for searching, but if there is no existing file by that name, then it is unspecified whether
47031 a regular file will be created. Likewise, if a non-directory file descriptor is successfully returned,
47032 it is unspecified whether that descriptor will have execute permissions as if by O_EXEC (note
47033 that it is unspecified whether O_EXEC and O_SEARCH have the same value).

47034 **RATIONALE**

47035 Some implementations permit opening FIFOs with O_RDWR. Since FIFOs could be
47036 implemented in other ways, and since two file descriptors can be used to the same effect, this
47037 possibility is left as undefined.

47038 See [getgroups\(\)](#) about the group of a newly created file.

47039 The use of *open()* to create a regular file is preferable to the use of *creat()*, because the latter is
47040 redundant and included only for historical reasons.

47041 The use of the O_TRUNC flag on FIFOs and directories (pipes cannot be *open()*-ed) must be
47042 permissible without unexpected side-effects (for example, *creat()* on a FIFO must not remove
47043 data). Since terminal special files might have type-ahead data stored in the buffer, O_TRUNC
47044 should not affect their content, particularly if a program that normally opens a regular file
47045 should open the current controlling terminal instead. Other file types, particularly
47046 implementation-defined ones, are left implementation-defined.

47047 POSIX.1-2017 permits [EACCES] to be returned for conditions other than those explicitly listed.

47048 The O_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a
47049 controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1-2017 does
47050 not specify how a controlling terminal is acquired, but it allows an implementation to provide
47051 this on *open()* if the O_NOCTTY flag is not set and other conditions specified in XBD [Chapter 11](#)
47052 (on page 199) are met.

47053 In historical implementations the value of O_RDONLY is zero. Because of that, it is not possible
47054 to detect the presence of O_RDONLY and another option. Future implementations should
47055 encode O_RDONLY and O_WRONLY as bit flags so that:

```
47056 O_RDONLY | O_WRONLY == O_RDWR
```

47057 O_EXEC and O_SEARCH are specified as two of the five file access modes. Since O_EXEC does
47058 not apply to directories, and O_SEARCH only applies to directories, their values need not be
47059 distinct. Since O_RDONLY has historically had the value zero, implementations are not able to
47060 distinguish between O_SEARCH and O_SEARCH | O_RDONLY, and similarly for O_EXEC.

47061 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,
47062 the *open()* function, when called with O_CREAT and O_EXCL, is required to fail with [EEXIST]
47063 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This
47064 behavior is required so that privileged applications can create a new file in a known location
47065 without the possibility that a symbolic link might cause the file to be created in a different
47066 location.

47067 For example, a privileged application that must create a file with a predictable name in a user-
47068 writable directory, such as the user's home directory, could be compromised if the user creates a
47069 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can
47070 influence the contents of a file, the user could compromise the system by creating a new system
47071 configuration or spool file that would then be interpreted by the system. The test for a symbolic
47072 link which refers to a nonexisting file must be atomic with the creation of a new file.

47073 In addition, the *open()* function refuses to open non-directories if the O_DIRECTORY flag is set.
47074 This avoids race conditions whereby a user might compromise the system by substituting a hard
47075 link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where
47076 opening a file even for read access might have undesirable side-effects.

47077 In addition, the *open()* function does not follow symbolic links if the O_NOFOLLOW flag is set.
47078 This avoids race conditions whereby a user might compromise the system by substituting a
47079 symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where
47080 opening a file even for read access might have undesirable side-effects.

47081 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
47082 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
47083 that implementations provide a way to have the group ID be set to the group ID of the
47084 containing directory, but did not prohibit implementations also supporting a way to set the
47085 group ID to the effective group ID of the creating process. Conforming applications should not
47086 assume which group ID will be used. If it matters, an application can use *chown()* to set the
47087 group ID after the file is created, or determine under what conditions the implementation will
47088 set the desired group ID.

47089 The purpose of the *openat()* function is to enable opening files in directories other than the
47090 current working directory without exposure to race conditions. Any part of the path of a file
47091 could be changed in parallel to a call to *open()*, resulting in unspecified behavior. By opening a
47092 file descriptor for the target directory and using the *openat()* function it can be guaranteed that
47093 the opened file is located relative to the desired directory. Some implementations use the
47094 *openat()* function for other purposes as well. In some cases, if the *oflag* parameter has the

47095 O_XATTR bit set, the returned file descriptor provides access to extended attributes. This
47096 functionality is not standardized here.

47097 FUTURE DIRECTIONS

47098 None.

47099 SEE ALSO

47100 *chmod(), close(), creat(), dirfd(), dup(), exec, fcntl(), fdopendir(), link(), lseek(), mkdtemp(),*
47101 *mknod(), read(), symlink(), umask(), unlockpt(), write()*

47102 XBD Chapter 11 (on page 199), [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)

47103 CHANGE HISTORY

47104 First released in Issue 1. Derived from Issue 1 of the SVID.

47105 Issue 5

47106 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
47107 Threads Extension.

47108 Large File Summit extensions are added.

47109 Issue 6

47110 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

47111 The following new requirements on POSIX implementations derive from alignment with the
47112 Single UNIX Specification:

47113 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
47114 required for conforming implementations of previous POSIX specifications, it was not
47115 required for UNIX applications.

47116 In the DESCRIPTION, O_CREAT is amended to state that the group ID of the file is set to
47117 the group ID of the file's parent directory or to the effective group ID of the process. This is
47118 a FIPS requirement.

47119 In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open
47120 file description. This change is to support large files.

47121 In the ERRORS section, the [Eoverflow] condition is added. This change is to support
47122 large files.

47123 The [ENXIO] mandatory error condition is added.

47124 The [EINVAL], [ENAMETOOLONG], and [ETXTBSY] optional error conditions are added.

47125 The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI
47126 STREAMS Option Group are marked.

47127 The following changes were made to align with the IEEE P1003.1a draft standard:

47128 An explanation is added of the effect of the O_CREAT and O_EXCL flags when the path
47129 refers to a symbolic link.

47130 The [ELOOP] optional error condition is added.

47131 The normative text is updated to avoid use of the term "must" for application requirements.

47132 The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

47133 **Issue 7**

- 47134 Austin Group Interpretations 1003.1-2001 #113 and #143 are applied.
- 47135 Austin Group Interpretation 1003.1-2001 #144 is applied, adding the `O_TTY_INIT` flag.
- 47136 Austin Group Interpretation 1003.1-2001 #171 is applied, adding support to set the
47137 `FD_CLOEXEC` flag atomically at `open()`, and adding the `F_DUPFD_CLOEXEC` flag.
- 47138 SD5-XBD-ERN-4 is applied, changing the definition of the `[EMFILE]` error.
- 47139 This page is revised and the `openat()` function is added from The Open Group Technical
47140 Standard, 2006, Extended API Set Part 2.
- 47141 Functionality relating to the XSI `STREAMS` option is marked obsolescent.
- 47142 Changes are made related to support for finegrained timestamps.
- 47143 Changes are made to allow a directory to be opened for searching.
- 47144 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0418 [292], XSH/TC1-2008/0419 [141],
47145 XSH/TC1-2008/0420 [461], XSH/TC1-2008/0421 [390], XSH/TC1-2008/0422 [146],
47146 XSH/TC1-2008/0423 [324], XSH/TC1-2008/0424 [292], XSH/TC1-2008/0425 [278],
47147 XSH/TC1-2008/0426 [278], XSH/TC1-2008/0427 [291], and XSH/TC1-2008/0428 [307] are
47148 applied.
- 47149 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0235 [873], XSH/TC2-2008/0236 [835],
47150 XSH/TC2-2008/0237 [847], XSH/TC2-2008/0238 [817], XSH/TC2-2008/0239 [835],
47151 XSH/TC2-2008/0240 [847], XSH/TC2-2008/0241 [822], XSH/TC2-2008/0242 [817], and
47152 XSH/TC2-2008/0243 [943] are applied.

47153 **NAME**

47154 open_memstream, open_wmemstream — open a dynamic memory buffer stream

47155 **SYNOPSIS**

```
47156 CX #include <stdio.h>
47157 FILE *open_memstream(char **bufp, size_t *sizep);
47158 #include <wchar.h>
47159 FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

47160 **DESCRIPTION**

47161 The *open_memstream()* and *open_wmemstream()* functions shall create an I/O stream associated
 47162 with a dynamically allocated memory buffer. The stream shall be opened for writing and shall
 47163 be seekable.

47164 The stream associated with a call to *open_memstream()* shall be byte-oriented.

47165 The stream associated with a call to *open_wmemstream()* shall be wide-oriented.

47166 The stream shall maintain a current position in the allocated buffer and a current buffer length.
 47167 The position shall be initially set to zero (the start of the buffer). Each write to the stream shall
 47168 start at the current position and move this position by the number of successfully written bytes
 47169 for *open_memstream()* or the number of successfully written wide characters for
 47170 *open_wmemstream()*. The length shall be initially set to zero. If a write moves the position to a
 47171 value larger than the current length, the current length shall be set to this position. In this case a
 47172 null character for *open_memstream()* or a null wide character for *open_wmemstream()* shall be
 47173 appended to the current buffer. For both functions the terminating null is not included in the
 47174 calculation of the buffer length.

47175 After a successful *fflush()* or *fclose()*, the pointer referenced by *bufp* shall contain the address of
 47176 the buffer, and the variable pointed to by *sizep* shall contain the smaller of the current buffer
 47177 length and the number of bytes for *open_memstream()*, or the number of wide characters for
 47178 *open_wmemstream()*, between the beginning of the buffer and the current file position indicator.

47179 After a successful *fflush()* the pointer referenced by *bufp* and the variable referenced by *sizep*
 47180 remain valid only until the next write operation on the stream or a call to *fclose()*.

47181 After a successful *fclose()*, the pointer referenced by *bufp* can be passed to *free()*.

47182 **RETURN VALUE**

47183 Upon successful completion, these functions shall return a pointer to the object controlling the
 47184 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

47185 **ERRORS**

47186 These functions shall fail if:

47187 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

47188 These functions may fail if:

47189 [EINVAL] *bufp* or *sizep* are NULL.

47190 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

47191 [ENOMEM] Memory for the stream or the buffer could not be allocated.

47192 **EXAMPLES**

```

47193     #include <stdio.h>
47194     #include <stdlib.h>

47195     int
47196     main (void)
47197     {
47198         FILE *stream;
47199         char *buf;
47200         size_t len;
47201         off_t eob;

47202         stream = open_memstream (&buf, &len);
47203         if (stream == NULL)
47204             /* handle error */ ;
47205         fprintf (stream, "hello my world");
47206         fflush (stream);
47207         printf ("buf=%s, len=%zu\n", buf, len);
47208         eob = ftello(stream);
47209         fseeko (stream, 0, SEEK_SET);
47210         fprintf (stream, "good-bye");
47211         fseeko (stream, eob, SEEK_SET);
47212         fclose (stream);
47213         printf ("buf=%s, len=%zu\n", buf, len);
47214         free (buf);
47215         return 0;
47216     }

```

47217 This program produces the following output:

```

47218     buf=hello my world, len=14
47219     buf=good-bye world, len=14

```

47220 **APPLICATION USAGE**

47221 The buffer created by these functions should be freed by the application after closing the stream,
 47222 by means of a call to *free()*.

47223 **RATIONALE**

47224 These functions are similar to *fmemopen()* except that the memory is always allocated
 47225 dynamically by the function, and the stream is opened only for output.

47226 **FUTURE DIRECTIONS**

47227 None.

47228 **SEE ALSO**

47229 *fclose()*, *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *free()*, *freopen()*

47230 XBD [<stdio.h>](#), [<wchar.h>](#)

47231 **CHANGE HISTORY**

47232 First released in Issue 7.

47233 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0244 [588] and XSH/TC2-2008/0245
 47234 [586] are applied.

47235 **NAME**

47236 openat — open file relative to directory file descriptor

47237 **SYNOPSIS**

47238 #include <fcntl.h>

47239 int openat(int *fd*, const char **path*, int *oflag*, ...);

47240 **DESCRIPTION**

47241 Refer to *open()*.

47242 **NAME**

47243 opendir — open directory associated with file descriptor

47244 **SYNOPSIS**

47245 #include <dirent.h>

47246 DIR *opendir(const char *dirname);

47247 **DESCRIPTION**47248 Refer to *fdopendir()*.

47249 **NAME**

47250 openlog ‡'open a connection to the logging facility

47251 **SYNOPSIS**

47252 XSI #include <syslog.h>

47253 void openlog(const char *ident, int logopt, int facility);

47254 **DESCRIPTION**

47255 Refer to *closelog()*.

47256 **NAME**

47257 optarg, opterr, optind, optopt ‡options parsing variables

47258 **SYNOPSIS**

47259 #include <unistd.h>

47260 extern char *optarg;

47261 extern int opterr, optind, optopt;

47262 **DESCRIPTION**47263 Refer to *getopt()*.

47264 **NAME**

47265 pathconf ‡get configurable pathname variables

47266 **SYNOPSIS**

47267 #include <unistd.h>

47268 long pathconf(const char *path, int name);

47269 **DESCRIPTION**

47270 Refer to *fpathconf()*.

47271 **NAME**

47272 pause — suspend the thread until a signal is received

47273 **SYNOPSIS**

47274 #include <unistd.h>

47275 int pause(void);

47276 **DESCRIPTION**47277 The *pause()* function shall suspend the calling thread until delivery of a signal whose action is
47278 either to execute a signal-catching function or to terminate the process.47279 If the action is to terminate the process, *pause()* shall not return.47280 If the action is to execute a signal-catching function, *pause()* shall return after the signal-catching
47281 function returns.47282 **RETURN VALUE**47283 Since *pause()* suspends thread execution indefinitely unless interrupted by a signal, there is no
47284 successful completion return value. A value of -1 shall be returned and *errno* set to indicate the
47285 error.47286 **ERRORS**47287 The *pause()* function shall fail if:47288 [EINTR] A signal is caught by the calling process and control is returned from the
47289 signal-catching function.47290 **EXAMPLES**

47291 None.

47292 **APPLICATION USAGE**47293 Many common uses of *pause()* have timing windows. The scenario involves checking a
47294 condition related to a signal and, if the signal has not occurred, calling *pause()*. When the signal
47295 occurs between the check and the call to *pause()*, the process often blocks indefinitely. The
47296 *sigprocmask()* and *sigsuspend()* functions can be used to avoid this type of problem.47297 **RATIONALE**

47298 None.

47299 **FUTURE DIRECTIONS**

47300 None.

47301 **SEE ALSO**47302 [sigsuspend\(\)](#)47303 XBD [<unistd.h>](#)47304 **CHANGE HISTORY**

47305 First released in Issue 1. Derived from Issue 1 of the SVID.

47306 **Issue 5**

47307 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

47308 **Issue 6**

47309 The APPLICATION USAGE section is added.

47310 **NAME**

47311 pclose — close a pipe stream to or from a process

47312 **SYNOPSIS**

```
47313 CX #include <stdio.h>
47314 int pclose(FILE *stream);
```

47315 **DESCRIPTION**

47316 The *pclose()* function shall close a stream that was opened by *popen()*, wait for the command to
 47317 terminate, and return the termination status of the process that was running the command
 47318 language interpreter. However, if a call caused the termination status to be unavailable to
 47319 *pclose()*, then *pclose()* shall return -1 with *errno* set to [ECHILD] to report this situation. This can
 47320 happen if the application calls one of the following functions:

47321 *wait()*47322 *waitpid()* with a *pid* argument less than or equal to 0 or equal to the process ID of the
47323 command line interpreter47324 Any other function not defined in this volume of POSIX.1-2017 that could do one of the
47325 above47326 In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.47327 If the command language interpreter cannot be executed, the child termination status returned
47328 by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or
47329 *_exit(127)*.47330 The *pclose()* function shall not affect the termination status of any child of the calling process
47331 other than the one created by *popen()* for the associated stream.47332 If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of
47333 *pclose()* is undefined.47334 If a thread is canceled during execution of *pclose()*, the behavior is undefined.47335 **RETURN VALUE**47336 Upon successful return, *pclose()* shall return the termination status of the command language
47337 interpreter. Otherwise, *pclose()* shall return -1 and set *errno* to indicate the error.47338 **ERRORS**47339 The *pclose()* function shall fail if:

47340 [ECHILD] The status of the child process could not be obtained, as described above.

47341 **EXAMPLES**

47342 None.

47343 **APPLICATION USAGE**

47344 None.

47345 **RATIONALE**

47346 There is a requirement that *pclose()* not return before the child process terminates. This is
 47347 intended to disallow implementations that return [EINTR] if a signal is received while waiting.
 47348 If *pclose()* returned before the child terminated, there would be no way for the application to
 47349 discover which child used to be associated with the stream, and it could not do the cleanup
 47350 itself.

47351 If the stream pointed to by *stream* was not created by *popen()*, historical implementations of

47352 *pclose()* return `-1` without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case,
 47353 POSIX.1-2017 makes the behavior unspecified. An application should not use *pclose()* to close
 47354 any stream that was not created by *popen()*.

47355 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,
 47356 and SIGHUP while waiting for the child process to terminate. Since this behavior is not
 47357 described for the *pclose()* function in POSIX.1-2017, such implementations are not conforming.
 47358 Also, some historical implementations return [EINTR] if a signal is received, even though the
 47359 child process has not terminated. Such implementations are also considered non-conforming.

47360 Consider, for example, an application that uses:

```
47361 popen("command", "r")
```

47362 to start *command*, which is part of the same application. The parent writes a prompt to its
 47363 standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child
 47364 reads the response from the user, does some transformation on the response (pathname
 47365 expansion, perhaps) and writes the result to its standard output. The parent process reads the
 47366 result from the pipe, does something with it, and prints another prompt. The cycle repeats.
 47367 Assuming that both processes do appropriate buffer flushing, this would be expected to work.

47368 To conform to POSIX.1-2017, *pclose()* must use *waitpid()*, or some similar function, instead of
 47369 *wait()*.

47370 The code sample below illustrates how the *pclose()* function might be implemented on a system
 47371 conforming to POSIX.1-2017.

```
47372 int pclose(FILE *stream)
47373 {
47374     int stat;
47375     pid_t pid;
47376
47377     pid = <pid for process created for stream by popen(>
47378         (void) fclose(stream);
47379     while (waitpid(pid, &stat, 0) == -1) {
47380         if (errno != EINTR){
47381             stat = -1;
47382             break;
47383         }
47384     }
47385     return(stat);
47386 }
```

47386 FUTURE DIRECTIONS

47387 None.

47388 SEE ALSO

47389 [fork\(\)](#), [popen\(\)](#), [wait\(\)](#)

47390 XBD [<stdio.h>](#)

47391 CHANGE HISTORY

47392 First released in Issue 1. Derived from Issue 1 of the SVID.

47393 Issue 7

47394 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0246 [632] is applied.

47395 **NAME**47396 `perror` — write error messages to standard error47397 **SYNOPSIS**47398 `#include <stdio.h>`47399 `void perror(const char *s);`47400 **DESCRIPTION**

47401 CX The functionality described on this reference page is aligned with the ISO C standard. Any
47402 conflict between the requirements described here and the ISO C standard is unintentional. This
47403 volume of POSIX.1-2017 defers to the ISO C standard.

47404 The `perror()` function shall map the error number accessed through the symbol `errno` to a
47405 language-dependent error message, which shall be written to the standard error stream as
47406 follows:

47407 First (if `s` is not a null pointer and the character pointed to by `s` is not the null byte), the
47408 string pointed to by `s` followed by a <colon> and a <space>.

47409 Then an error message string followed by a <newline>.

47410 The contents of the error message strings shall be the same as those returned by `strerror()` with
47411 argument `errno`.

47412 CX The `perror()` function shall mark for update the last data modification and last file status change
47413 timestamps of the file associated with the standard error stream at some time between its
47414 successful completion and `exit()`, `abort()`, or the completion of `fflush()` or `fclose()` on `stderr`.

47415 The `perror()` function shall not change the orientation of the standard error stream.

47416 On error, `perror()` shall set the error indicator for the stream to which `stderr` points, and shall set
47417 `errno` to indicate the error.

47418 Since no value is returned, an application wishing to check for error situations should call
47419 `clearerr(stderr)` before calling `perror()`, then if `ferror(stderr)` returns non-zero, the value of `errno`
47420 indicates which error occurred.

47421 **RETURN VALUE**

47422 The `perror()` function shall not return a value.

47423 **ERRORS**

47424 CX Refer to [fputc\(\)](#).

47425 **EXAMPLES**47426 **Printing an Error Message for a Function**

47427 The following example replaces `bufptr` with a buffer that is the necessary size. If an error occurs,
47428 the `perror()` function prints a message and the program exits.

```
47429 #include <stdio.h>
47430 #include <stdlib.h>
47431 ...
47432 char *bufptr;
47433 size_t szbuf;
47434 ...
47435 if ((bufptr = malloc(szbuf)) == NULL) {
47436     perror("malloc"); exit(2);
47437 }
```

47438 . . .

47439 **APPLICATION USAGE**

47440 Application writers may prefer to use alternative interfaces instead of *perror()*, such as

47441 *strerror_r()* in combination with *fprintf()*.

47442 **RATIONALE**

47443 None.

47444 **FUTURE DIRECTIONS**

47445 None.

47446 **SEE ALSO**

47447 *fprintf()*, *fputc()*, *psiginfo()*, *strerror()*

47448 XBD <stdio.h>

47449 **CHANGE HISTORY**

47450 First released in Issue 1. Derived from Issue 1 of the SVID.

47451 **Issue 5**

47452 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the

47453 orientation of the standard error stream.

47454 **Issue 6**

47455 Extensions beyond the ISO C standard are marked.

47456 **Issue 7**

47457 SD5-XSH-ERN-95 is applied.

47458 Changes are made related to support for finegrained timestamps.

47459 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0429 [389,401], XSH/TC1-2008/0430

47460 [389], and XSH/TC1-2008/0431 [389,401] are applied.

47461 **NAME**

47462 pipe — create an interprocess channel

47463 **SYNOPSIS**

47464 #include <unistd.h>

47465 int pipe(int *fildes*[2]);47466 **DESCRIPTION**

47467 The *pipe()* function shall create a pipe and place two file descriptors, one each into the
 47468 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write
 47469 ends of the pipe. The file descriptors shall be allocated as described in [Section 2.14](#) (on page 549).
 47470 The O_NONBLOCK and FD_CLOEXEC flags shall be clear on both file descriptors. (The *fcntl()*
 47471 function can be used to set both these flags.)

47472 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A
 47473 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a
 47474 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether
 47475 *fildes*[1] is also open for reading.

47476 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open
 47477 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

47478 The pipe's user ID shall be set to the effective user ID of the calling process.

47479 The pipe's group ID shall be set to the effective group ID of the calling process.

47480 Upon successful completion, *pipe()* shall mark for update the last data access, last data
 47481 modification, and last file status change timestamps of the pipe.

47482 **RETURN VALUE**

47483 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 47484 indicate the error, no file descriptors shall be allocated and the contents of *fildes* shall be left
 47485 unmodified.

47486 **ERRORS**47487 The *pipe()* function shall fail if:

47488 [EMFILE] All, or all but one, of the file descriptors available to the process are currently
 47489 open.

47490 [ENFILE] The number of simultaneously open files in the system would exceed a
 47491 system-imposed limit.

47492 **EXAMPLES**47493 **Using a Pipe to Pass Data Between a Parent Process and a Child Process**

47494 The following example demonstrates the use of a pipe to transfer data between a parent process
 47495 and a child process. Error handling is excluded, but otherwise this code demonstrates good
 47496 practice when using pipes: after the *fork()* the two processes close the unused ends of the pipe
 47497 before they commence transferring data.

47498 #include <stdlib.h>

47499 #include <unistd.h>

47500 ...

47501 int fildes[2];

47502 const int BSIZE = 100;

47503 char buf[BSIZE];

```

47504     ssize_t nbytes;
47505     int status;

47506     status = pipe(fildes);
47507     if (status == -1 ) {
47508         /* an error occurred */
47509         ...
47510     }

47511     switch (fork()) {
47512     case -1: /* Handle error */
47513         break;

47514     case 0: /* Child - reads from pipe */
47515         close(fildes[1]); /* Write end is unused */
47516         nbytes = read(fildes[0], buf, BSIZE); /* Get data from pipe */
47517         /* At this point, a further read would see end-of-file ... */
47518         close(fildes[0]); /* Finished with pipe */
47519         exit(EXIT_SUCCESS);

47520     default: /* Parent - writes to pipe */
47521         close(fildes[0]); /* Read end is unused */
47522         write(fildes[1], "Hello world\n", 12); /* Write data on pipe */
47523         close(fildes[1]); /* Child will see EOF */
47524         exit(EXIT_SUCCESS);
47525     }

```

APPLICATION USAGE

47526 None.

RATIONALE

47529 The wording carefully avoids using the verb “to open” in order to avoid any implication of use
 47530 of *open()*; see also *write()*.

FUTURE DIRECTIONS

47531 None.

SEE ALSO

47534 [Section 2.14](#) (on page 549), *fcntl()*, *read()*, *write()*

47535 XBD [<fcntl.h>](#), [<unistd.h>](#)

CHANGE HISTORY

47537 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

47539 The following new requirements on POSIX implementations derive from alignment with the
 47540 Single UNIX Specification:

47541 The DESCRIPTION is updated to indicate that certain dispositions of *fildes[0]* and *fildes[1]*
 47542 are unspecified.

47543 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/65 is applied, adding the example to the
 47544 EXAMPLES section.

47545 **Issue 7**

47546 SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe's user
47547 ID and group ID.

47548 Changes are made related to support for finegrained timestamps.

47549 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0247 [835] and XSH/TC2-2008/0248
47550 [467,835] are applied.

47551 **NAME**

47552 poll ¶input/output multiplexing

47553 **SYNOPSIS**

47554 #include <poll.h>

47555 int poll(struct pollfd *fds*[], nfd_t *nfds*, int *timeout*);47556 **DESCRIPTION**

47557 The *poll()* function provides applications with a mechanism for multiplexing input/output over
 47558 a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* shall examine the
 47559 given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the
 47560 *fds* array is specified by *nfds*. The *poll()* function shall identify those file descriptors on which an
 47561 application can read or write data, or on which certain events have occurred.

47562 The *fds* argument specifies the file descriptors to be examined and the events of interest for each
 47563 file descriptor. It is a pointer to an array with one member for each open file descriptor of
 47564 interest. The array's members are **pollfd** structures within which *fd* specifies an open file
 47565 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the
 47566 following event flags:

47567	POLLIN	Data other than high-priority data may be read without blocking.
47568	OB XSR	For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
47569		This flag shall be equivalent to POLLRDNORM POLLRDBAND.
47570	POLLRDNORM	Normal data may be read without blocking.
47571	OB XSR	For STREAMS, data on priority band 0 may be read without blocking. This
47572		flag is set in <i>revents</i> even if the message is of zero length.
47573	POLLRDBAND	Priority data may be read without blocking.
47574	OB XSR	For STREAMS, data on priority bands greater than 0 may be read without
47575		blocking. This flag is set in <i>revents</i> even if the message is of zero length.
47576	POLLPRI	High-priority data may be read without blocking.
47577	OB XSR	For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
47578	POLLOUT	Normal data may be written without blocking.
47579	OB XSR	For STREAMS, data on priority band 0 may be written without blocking.
47580	POLLWRNORM	Equivalent to POLLOUT.
47581	POLLWRBAND	Priority data may be written.
47582	OB XSR	For STREAMS, data on priority bands greater than 0 may be written without
47583		blocking. If any priority band has been written to on this STREAM, this event
47584		only examines bands that have been written to at least once.
47585	POLLERR	An error has occurred on the device or stream. This flag is only valid in the
47586		<i>revents</i> bitmask; it shall be ignored in the <i>events</i> member.
47587	POLLHUP	A device has been disconnected, or a pipe or FIFO has been closed by the last
47588		process that had it open for writing. Once set, the hangup state of a FIFO shall
47589		persist until some process opens the FIFO for writing or until all read-only file
47590		descriptors for the FIFO are closed. This event and POLLOUT are mutually-
47591		exclusive; a stream can never be writable if a hangup has occurred. However,
47592		this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are
47593		not mutually-exclusive. This flag is only valid in the <i>revents</i> bitmask; it shall be

47594 ignored in the *events* member.

47595 POLLNVAL The specified *fd* value is invalid. This flag is only valid in the *revents* member;
47596 it shall ignored in the *events* member.

47597 The significance and semantics of normal, priority, and high-priority data are file and device-
47598 specific.

47599 If the value of *fd* is less than 0, *events* shall be ignored, and *revents* shall be set to 0 in that entry on
47600 return from *poll()*.

47601 In each **pollfd** structure, *poll()* shall clear the *revents* member, except that where the application
47602 requested a report on a condition by setting one of the bits of *events* listed above, *poll()* shall set
47603 the corresponding bit in *revents* if the requested condition is true. In addition, *poll()* shall set the
47604 POLLHUP, POLLERR, and POLLNVAL flag in *revents* if the condition is true, even if the
47605 application did not set the corresponding bit in *events*.

47606 If none of the defined events have occurred on any selected file descriptor, *poll()* shall wait at
47607 least *timeout* milliseconds for an event to occur on any of the selected file descriptors. If the value
47608 of *timeout* is 0, *poll()* shall return immediately. If the value of *timeout* is -1, *poll()* shall block until
47609 a requested event occurs or until the call is interrupted.

47610 Implementations may place limitations on the granularity of timeout intervals. If the requested
47611 timeout interval requires a finer granularity than the implementation supports, the actual
47612 timeout interval shall be rounded up to the next supported value.

47613 The *poll()* function shall not be affected by the O_NONBLOCK flag.

47614 The *poll()* function shall support regular files, terminal and pseudo-terminal devices, FIFOs,
47615 OB XSR pipes, sockets and STREAMS-based files. The behavior of *poll()* on elements of *fds* that refer to
47616 other types of file is unspecified.

47617 Regular files shall always poll TRUE for reading and writing.

47618 A file descriptor for a socket that is listening for connections shall indicate that it is ready for
47619 reading, once connections are available. A file descriptor for a socket that is connecting
47620 asynchronously shall indicate that it is ready for writing, once a connection has been established.

47621 Provided the application does not perform any action that results in unspecified or undefined
47622 behavior, the value of the *fd* and *events* members of each element of *fds* shall not be modified by
47623 *poll()*.

47624 **RETURN VALUE**

47625 Upon successful completion, *poll()* shall return a non-negative value. A positive value indicates
47626 the total number of **pollfd** structures that have selected events (that is, those for which the
47627 *revents* member is non-zero). A value of 0 indicates that the call timed out and no file descriptors
47628 have been selected. Upon failure, *poll()* shall return -1 and set *errno* to indicate the error.

47629 **ERRORS**

47630 The *poll()* function shall fail if:

47631 [EAGAIN] The allocation of internal data structures failed but a subsequent request may
47632 succeed.

47633 [EINTR] A signal was caught during *poll()*.

47634 OB XSR [EINVAL] The *nfds* argument is greater than {OPEN_MAX}, or one of the *fd* members
47635 refers to a STREAM or multiplexer that is linked (directly or indirectly)
47636 downstream from a multiplexer.

47637 **EXAMPLES**47638 **Checking for Events on a Stream**

47639 The following example opens a pair of STREAMS devices and then waits for either one to
47640 become writable. This example proceeds as follows:

- 47641 1. Sets the *timeout* parameter to 500 milliseconds.
- 47642 2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying
47643 POLLOUT and POLLWRBAND as the events of interest.
- 47644 The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how
47645 STREAMS devices can be named; STREAMS naming conventions may vary among
47646 systems conforming to the POSIX.1-2017.
- 47647 3. Uses the *ret* variable to determine whether an event has occurred on either of the two
47648 STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it
47649 has not occurred prior to the *poll()* call).
- 47650 4. Checks the returned value of *ret*. If a positive value is returned, one of the following can
47651 be done:
 - 47652 a. Priority data can be written to the open STREAM on priority bands greater than 0,
47653 because the POLLWRBAND event occurred on the open STREAM (*fds[0]* or *fds[1]*).
 - 47654 b. Data can be written to the open STREAM on priority-band 0, because the
47655 POLLOUT event occurred on the open STREAM (*fds[0]* or *fds[1]*).
- 47656 5. If the returned value is not a positive value, permission to write data to the open
47657 STREAM (on any priority band) is denied.
- 47658 6. If the POLLHUP event occurs on the open STREAM (*fds[0]* or *fds[1]*), the device on the
47659 open STREAM has disconnected.

```

47660 #include <stropts.h>
47661 #include <poll.h>
47662 ...
47663 struct pollfd fds[2];
47664 int timeout_msecs = 500;
47665 int ret;
47666     int i;

47667 /* Open STREAMS device. */
47668 fds[0].fd = open("/dev/dev0", ...);
47669 fds[1].fd = open("/dev/dev1", ...);
47670 fds[0].events = POLLOUT | POLLWRBAND;
47671 fds[1].events = POLLOUT | POLLWRBAND;

47672 ret = poll(fds, 2, timeout_msecs);

47673 if (ret > 0) {
47674     /* An event on one of the fds has occurred. */
47675     for (i=0; i<2; i++) {
47676         if (fds[i].revents & POLLWRBAND) {
47677             /* Priority data may be written on device number i. */
47678             ...
47679         }

```

```

47680         if (fds[i].revents & POLLOUT) {
47681             /* Data may be written on device number i. */
47682         ...
47683         }
47684         if (fds[i].revents & POLLHUP) {
47685             /* A hangup has occurred on device number i. */
47686         ...
47687         }
47688     }
47689 }

```

APPLICATION USAGE

47690 None.

RATIONALE

47693 The POLLHUP event does not occur for FIFOs just because the FIFO is not open for writing. It
 47694 only occurs when the FIFO is closed by the last writer and persists until some process opens the
 47695 FIFO for writing or until all read-only file descriptors for the FIFO are closed.

FUTURE DIRECTIONS

47696 None.

SEE ALSO

47699 [Section 2.6](#) (on page 500), [getmsg\(\)](#), [pselect\(\)](#), [putmsg\(\)](#), [read\(\)](#), [write\(\)](#)
 47700 XBD [<poll.h>](#), [<stropts.h>](#)

CHANGE HISTORY

47702 First released in Issue 4, Version 2.

Issue 5

47704 Moved from X/OPEN UNIX extension to BASE.

47705 The description of POLLWRBAND is updated.

Issue 6

47707 Text referring to sockets is added to the DESCRIPTION.

47708 Functionality relating to the XSI STREAMS Option Group is marked.

47709 The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of
 47710 POLLWRBAND.

47711 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/66 is applied, correcting the spacing in
 47712 the EXAMPLES section.

Issue 7

47714 Austin Group Interpretation 1003.1-2001 #209 is applied, clarifying the POLLHUP event.

47715 The *poll()* function is moved from the XSI option to the Base.

47716 Functionality relating to the XSI STREAMS option is marked obsolescent.

47717 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0249 [623] and XSH/TC2-2008/0250
 47718 [683] are applied.

47719 **NAME**

47720 popen — initiate pipe streams to or from a process

47721 **SYNOPSIS**

```
47722 CX #include <stdio.h>
47723 FILE *popen(const char *command, const char *mode);
```

47724 **DESCRIPTION**

47725 The *popen()* function shall execute the command specified by the string *command*. It shall create
 47726 a pipe between the calling program and the executed command, and shall return a pointer to a
 47727 stream that can be used to either read from or write to the pipe.

47728 The environment of the executed command shall be as if a child process were created within the
 47729 *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

```
47730 execl(shell path, "sh", "-c", command, (char *)0);
```

47731 where *shell path* is an unspecified pathname for the *sh* utility.

47732 The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open
 47733 in the parent process are closed in the new child process.

47734 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 47735 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be
 47736 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,
 47737 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the
 47738 pipe.
- 47739 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be
 47740 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,
 47741 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the
 47742 pipe.
- 47743 3. If *mode* is any other value, the result is unspecified.

47744 After *popen()*, both the parent and the child process shall be capable of executing independently
 47745 before either terminates.

47746 Pipe streams are byte-oriented.

47747 **RETURN VALUE**

47748 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to
 47749 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate
 47750 the error.

47751 **ERRORS**

47752 The *popen()* function shall fail if:

47753 [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

47754 The *popen()* function may fail if:

47755 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

47756 [EINVAL] The *mode* argument is invalid.

47757 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

47758 **EXAMPLES**47759 **Using popen() to Obtain a List of Files from the ls Utility**

47760 The following example demonstrates the use of *popen()* and *pclose()* to execute the command *ls**
 47761 in order to obtain a list of files in the current directory:

```

47762 #include <stdio.h>
47763 ...
47764 FILE *fp;
47765 int status;
47766 char path[PATH_MAX];
47767 fp = popen("ls *", "r");
47768 if (fp == NULL)
47769     /* Handle error */;
47770 while (fgets(path, PATH_MAX, fp) != NULL)
47771     printf("%s", path);
47772 status = pclose(fp);
47773 if (status == -1) {
47774     /* Error reported by pclose() */
47775     ...
47776 } else {
47777     /* Use macros described under wait() to inspect `status' in order
47778        to determine success/failure of command executed by popen() */
47779     ...
47780 }

```

47781 **APPLICATION USAGE**

47782 Since open files are shared, a mode *r* command can be used as an input filter and a mode *w*
 47783 command as an output filter.

47784 Buffered reading before opening an input filter may leave the standard input of that filter
 47785 mispositioned. Similar problems with an output filter may be prevented by careful buffer
 47786 flushing; for example, with *fflush()*.

47787 A stream opened by *popen()* should be closed by *pclose()*.

47788 The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and
 47789 *wb* might be supported by specific implementations, but these would not be portable features.
 47790 Note that historical implementations of *popen()* only check to see if the first character of *mode* is
 47791 *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be
 47792 treated as *mode w*.

47793 If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a
 47794 stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process
 47795 started by *popen()*.

47796 To determine whether or not the environment specified in the Shell and Utilities volume of
 47797 POSIX.1-2017 is present, use the function call:

```
47798 sysconf(_SC_2_VERSION)
```

47799 (See *sysconf()*).

47800 **RATIONALE**

47801 The *popen()* function should not be used by programs that have set user (or group) ID privileges.
47802 The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead.
47803 This prevents any unforeseen manipulation of the environment of the user that could cause
47804 execution of commands not anticipated by the calling program.

47805 If the original and *popen()*ed processes both intend to read or write or read and write a common
47806 file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for
47807 sharing file handles must be observed (see [Section 2.5.1](#), on page 497).

47808 **FUTURE DIRECTIONS**

47809 None.

47810 **SEE ALSO**

47811 [Section 2.5](#) (on page 495), *fork()*, *pclose()*, *pipe()*, *sysconf()*, *system()*, *wait()*, *waitid()*

47812 XBD [<stdio.h>](#)

47813 XCU *sh*

47814 **CHANGE HISTORY**

47815 First released in Issue 1. Derived from Issue 1 of the SVID.

47816 **Issue 5**

47817 A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

47818 **Issue 6**

47819 The following new requirements on POSIX implementations derive from alignment with the
47820 Single UNIX Specification:

47821 The optional [EMFILE] error condition is added.

47822 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/67 is applied, adding the example to the
47823 EXAMPLES section.

47824 **Issue 7**

47825 Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the
47826 DESCRIPTION.

47827 SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a
47828 “may fail” to a “shall fail”.

47829 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0432 [14] is applied.

47830 **NAME**

47831 posix_fadvise ‡file advisory information (ADVANCED REALTIME)

47832 **SYNOPSIS**

```
47833 ADV #include <fcntl.h>
47834 int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

47835 **DESCRIPTION**

47836 The *posix_fadvise()* function shall advise the implementation on the expected behavior of the
 47837 application with respect to the data in the file associated with the open file descriptor, *fd*, starting
 47838 at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len*
 47839 is zero, all data following *offset* is specified. The implementation may use this information to
 47840 optimize handling of the specified data. The *posix_fadvise()* function shall have no effect on the
 47841 semantics of other operations on the specified data, although it may affect the performance of
 47842 other operations.

47843 The advice to be applied to the data is specified by the *advice* parameter and may be one of the
 47844 following values:

47845 POSIX_FADV_NORMAL

47846 Specifies that the application has no advice to give on its behavior with respect to the
 47847 specified data. It is the default characteristic if no advice is given for an open file.

47848 POSIX_FADV_SEQUENTIAL

47849 Specifies that the application expects to access the specified data sequentially from lower
 47850 offsets to higher offsets.

47851 POSIX_FADV_RANDOM

47852 Specifies that the application expects to access the specified data in a random order.

47853 POSIX_FADV_WILLNEED

47854 Specifies that the application expects to access the specified data in the near future.

47855 POSIX_FADV_DONTNEED

47856 Specifies that the application expects that it will not access the specified data in the near
 47857 future.

47858 POSIX_FADV_NOREUSE

47859 Specifies that the application expects to access the specified data once and then not reuse it
 47860 thereafter.

47861 These values are defined in **<fcntl.h>**.

47862 **RETURN VALUE**

47863 Upon successful completion, *posix_fadvise()* shall return zero; otherwise, an error number shall
 47864 be returned to indicate the error.

47865 **ERRORS**

47866 The *posix_fadvise()* function shall fail if:

- 47867 [EBADF] The *fd* argument is not a valid file descriptor.
- 47868 [EINVAL] The value of *advice* is invalid, or the value of *len* is less than zero.
- 47869 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.

47870 **EXAMPLES**

47871 None.

47872 **APPLICATION USAGE**

47873 The *posix_fadvise()* function is part of the Advisory Information option and need not be
47874 provided on all implementations.

47875 **RATIONALE**

47876 None.

47877 **FUTURE DIRECTIONS**

47878 None.

47879 **SEE ALSO**47880 *posix_madvise()*

47881 XBD <fcntl.h>

47882 **CHANGE HISTORY**

47883 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

47884 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

47885 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/68 is applied, changing the function
47886 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the
47887 standard developers felt it acceptable to make this change before implementations of this
47888 function become widespread.

47889 **Issue 7**

47890 Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the
47891 [EINVAL] error.

47892 **NAME**

47893 posix_fallocate — file space control (**ADVANCED REALTIME**)

47894 **SYNOPSIS**

```
47895 ADV #include <fcntl.h>
47896 int posix_fallocate(int fd, off_t offset, off_t len);
```

47897 **DESCRIPTION**

47898 The *posix_fallocate()* function shall ensure that any required storage for regular file data starting
 47899 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If
 47900 *posix_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail due
 47901 to the lack of free space on the file system storage media.

47902 If the *offset+len* is beyond the current file size, then *posix_fallocate()* shall adjust the file size to
 47903 *offset+len*. Otherwise, the file size shall not be changed.

47904 It is implementation-defined whether a previous *posix_fadvise()* call influences allocation
 47905 strategy.

47906 Space allocated via *posix_fallocate()* shall be freed by a successful call to *creat()* or *open()* that
 47907 truncates the size of the file. Space allocated via *posix_fallocate()* may be freed by a successful call
 47908 to *truncate()* that reduces the file size to a size smaller than *offset+len*.

47909 **RETURN VALUE**

47910 Upon successful completion, *posix_fallocate()* shall return zero; otherwise, an error number shall
 47911 be returned to indicate the error.

47912 **ERRORS**

47913 The *posix_fallocate()* function shall fail if:

- 47914 [EBADF] The *fd* argument is not a valid file descriptor.
 - 47915 [EBADF] The *fd* argument references a file that was opened without write permission.
 - 47916 [EFBIG] The value of *offset+len* is greater than the maximum file size.
 - 47917 [EINTR] A signal was caught during execution.
 - 47918 [EINVAL] The *len* argument is less than zero, or the *offset* argument is less than zero, or
 47919 the underlying file system does not support this operation.
 - 47920 [EIO] An I/O error occurred while reading from or writing to a file system.
 - 47921 [ENODEV] The *fd* argument does not refer to a regular file.
 - 47922 [ENOSPC] There is insufficient free space remaining on the file system storage media.
 - 47923 [ESPIPE] The *fd* argument is associated with a pipe or FIFO.
- 47924 The *posix_fallocate()* function may fail if:
- 47925 [EINVAL] The *len* argument is zero.

47926 **EXAMPLES**

47927 None.

47928 **APPLICATION USAGE**47929 The *posix_fallocate()* function is part of the Advisory Information option and need not be
47930 provided on all implementations.47931 **RATIONALE**

47932 None.

47933 **FUTURE DIRECTIONS**

47934 None.

47935 **SEE ALSO**47936 *creat()*, *fruncate()*, *open()*, *unlink()*47937 XBD <[fcntl.h](#)>47938 **CHANGE HISTORY**

47939 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

47940 In the SYNOPSIS, the inclusion of <[sys/types.h](#)> is no longer required.47941 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/69 is applied, changing the function
47942 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the
47943 standard developers felt it acceptable to make this change before implementations of this
47944 function become widespread.47945 **Issue 7**47946 Austin Group Interpretations 1003.1-2001 #022, #024, and #162 are applied, changing the
47947 definition of the [EINVAL] error.

47948 **NAME**

47949 posix_madvise — memory advisory information and alignment control (**ADVANCED**
47950 **REALTIME**)

47951 **SYNOPSIS**

```
47952 ADV #include <sys/mman.h>
47953 int posix_madvise(void *addr, size_t len, int advice);
```

47954 **DESCRIPTION**

47955 The *posix_madvise()* function shall advise the implementation on the expected behavior of the
47956 application with respect to the data in the memory starting at address *addr*, and continuing for
47957 *len* bytes. The implementation may use this information to optimize handling of the specified
47958 data. The *posix_madvise()* function shall have no effect on the semantics of access to memory in
47959 the specified range, although it may affect the performance of access.

47960 The implementation may require that *addr* be a multiple of the page size, which is the value
47961 returned by *sysconf()* when the name value *_SC_PAGESIZE* is used.

47962 The advice to be applied to the memory range is specified by the *advice* parameter and may be
47963 one of the following values:

47964 **POSIX_MADV_NORMAL**
47965 Specifies that the application has no advice to give on its behavior with respect to the
47966 specified range. It is the default characteristic if no advice is given for a range of memory.

47967 **POSIX_MADV_SEQUENTIAL**
47968 Specifies that the application expects to access the specified range sequentially from lower
47969 addresses to higher addresses.

47970 **POSIX_MADV_RANDOM**
47971 Specifies that the application expects to access the specified range in a random order.

47972 **POSIX_MADV_WILLNEED**
47973 Specifies that the application expects to access the specified range in the near future.

47974 **POSIX_MADV_DONTNEED**
47975 Specifies that the application expects that it will not access the specified range in the near
47976 future.

47977 These values are defined in the **<sys/mman.h>** header.

47978 **RETURN VALUE**

47979 Upon successful completion, *posix_madvise()* shall return zero; otherwise, an error number shall
47980 be returned to indicate the error.

47981 **ERRORS**

47982 The *posix_madvise()* function shall fail if:

- 47983 [EINVAL] The value of *advice* is invalid.
- 47984 [ENOMEM] Addresses in the range starting at *addr* and continuing for *len* bytes are partly
47985 or completely outside the range allowed for the address space of the calling
47986 process.

- 47987 The *posix_madvise()* function may fail if:
- 47988 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the
47989 name value *_SC_PAGESIZE* is used.
- 47990 [EINVAL] The value of *len* is zero.
- 47991 **EXAMPLES**
- 47992 None.
- 47993 **APPLICATION USAGE**
- 47994 The *posix_madvise()* function is part of the Advisory Information option and need not be
47995 provided on all implementations.
- 47996 **RATIONALE**
- 47997 None.
- 47998 **FUTURE DIRECTIONS**
- 47999 None.
- 48000 **SEE ALSO**
- 48001 *mmap()*, *posix_fadvise()*, *sysconf()*
- 48002 XBD <[sys/mman.h](#)>
- 48003 **CHANGE HISTORY**
- 48004 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
- 48005 IEEE PASC Interpretation 1003.1 #102 is applied.

48006 **NAME**

48007 posix_mem_offset ¶ find offset and length of a mapped typed memory block (**ADVANCED**
48008 **REALTIME**)

48009 **SYNOPSIS**

```
48010 TYM    #include <sys/mman.h>
48011        int posix_mem_offset(const void *restrict addr, size_t len,
48012                            off_t *restrict off, size_t *restrict contig_len,
48013                            int *restrict fildes);
```

48014 **DESCRIPTION**

48015 The *posix_mem_offset()* function shall return in the variable pointed to by *off* a value that
48016 identifies the offset (or location), within a memory object, of the memory block currently
48017 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used
48018 (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since
48019 the mapping was established, the returned value of *fildes* shall be -1 . The *len* argument specifies
48020 the length of the block of the memory object the user wishes the offset for; upon return, the
48021 value pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block
48022 of the memory object that is currently mapped to the calling process starting at *addr*, whichever
48023 is smaller.

48024 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len*
48025 values obtained by calling *posix_mem_offset()* are used in a call to *mmap()* with a file descriptor
48026 that refers to the same memory pool as *fildes* (either through the same port or through a different
48027 port), and that was opened with neither the `POSIX_TYPED_MEM_ALLOCATE` nor the
48028 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the typed memory area that is mapped shall
48029 be exactly the same area that was mapped at *addr* in the address space of the process that called
48030 *posix_mem_offset()*.

48031 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this
48032 function is implementation-defined.

48033 **RETURN VALUE**

48034 Upon successful completion, the *posix_mem_offset()* function shall return zero; otherwise, the
48035 corresponding error status value shall be returned.

48036 **ERRORS**

48037 The *posix_mem_offset()* function shall fail if:

48038 [EACCES] The process has not mapped a memory object supported by this function at
48039 the given address *addr*.

48040 This function shall not return an error code of [EINTR].

48041 **EXAMPLES**

48042 None.

48043 **APPLICATION USAGE**

48044 None.

48045 **RATIONALE**

48046 None.

48047 **FUTURE DIRECTIONS**

48048 None.

48049 **SEE ALSO**

48050 *mmap()*, *posix_typed_mem_open()*

48051 XBD <[sys/mman.h](#)>

48052 **CHANGE HISTORY**

48053 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

48054 **NAME**

48055 posix_memalign ¶aligned memory allocation (ADVANCED REALTIME)

48056 **SYNOPSIS**

```
48057 ADV #include <stdlib.h>
48058 int posix_memalign(void **memptr, size_t alignment, size_t size);
```

48059 **DESCRIPTION**

48060 The *posix_memalign()* function shall allocate *size* bytes aligned on a boundary specified by
 48061 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*
 48062 shall be a power of two multiple of *sizeof(void *)*.

48063 Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

48064 If the size of the space requested is 0, the behavior is implementation-defined: either a null
 48065 pointer shall be returned in *memptr*, or the behavior shall be as if the size were some non-zero
 48066 value, except that the behavior is undefined if the the value returned in *memptr* is used to access
 48067 an object.

48068 CX The *free()* function shall deallocate memory that has previously been allocated by
 48069 *posix_memalign()*.

48070 **RETURN VALUE**

48071 Upon successful completion, *posix_memalign()* shall return zero; otherwise, an error number
 48072 shall be returned to indicate the error and the contents of *memptr* shall either be left unmodified
 48073 or be set to a null pointer.

48074 If *size* is 0, either:

48075 *posix_memalign()* shall not attempt to allocate any space, in which case either an
 48076 implementation-defined error number shall be returned, or zero shall be returned with a
 48077 null pointer returned in *memptr*, or

48078 *posix_memalign()* shall attempt to allocate some space and, if the allocation succeeds, zero
 48079 shall be returned and a pointer to the allocated space shall be returned in *memptr*. The
 48080 application shall ensure that the pointer is not used to access an object.

48081 **ERRORS**

48082 The *posix_memalign()* function shall fail if:

48083 [EINVAL] The value of the alignment parameter is not a power of two multiple of
 48084 *sizeof(void *)*.

48085 [ENOMEM] There is insufficient memory available with the requested alignment.

48086 **EXAMPLES**

48087 The following example shows how applications can obtain consistent behavior on error by
 48088 setting **memptr* to be a null pointer before calling *posix_memalign()*.

```
48089 void *ptr = NULL;
48090 ...
48091 //do some work, which might goto error
48092 if (posix_memalign(&ptr, align, size))
48093     goto error;
48094
48095 //do some more work, which might goto error
48096 ...
48097 error:
```

```
48097         free(ptr);  
48098         //more cleanup;
```

48099 APPLICATION USAGE

48100 The *posix_memalign()* function is part of the Advisory Information option and need not be
48101 provided on all implementations.

48102 RATIONALE

48103 None.

48104 FUTURE DIRECTIONS

48105 None.

48106 SEE ALSO

48107 *free()*, *malloc()*

48108 XBD <stdlib.h>

48109 CHANGE HISTORY

48110 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48111 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

48112 Issue 7

48113 Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment*
48114 argument in the DESCRIPTION.

48115 Austin Group Interpretation 1003.1-2001 #152 is applied, clarifying the behavior when the size of
48116 the space requested is 0.

48117 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0251 [526] and XSH/TC2-2008/0252
48118 [520,526] are applied.

48119 **NAME**

48120 posix_openpt ‡open a pseudo-terminal device

48121 **SYNOPSIS**

```
48122 XSI #include <stdlib.h>
48123 #include <fcntl.h>
48124 int posix_openpt(int oflag);
```

48125 **DESCRIPTION**

48126 The *posix_openpt()* function shall establish a connection between a master device for a pseudo-
 48127 terminal and a file descriptor. The file descriptor shall be allocated as described in [Section 2.14](#)
 48128 (on page 549) and can be used by other I/O functions that refer to that pseudo-terminal.

48129 The file status flags and file access modes of the open file description shall be set according to
 48130 the value of *oflag*.

48131 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 48132 in **<fcntl.h>**:

- 48133 O_RDWR Open for reading and writing.
- 48134 O_NOCTTY If set *posix_openpt()* shall not cause the terminal device to become the
 48135 controlling terminal for the process.

48136 The behavior of other values for the *oflag* argument is unspecified.

48137 **RETURN VALUE**

48138 Upon successful completion, the *posix_openpt()* function shall open a file descriptor for a master
 48139 pseudo-terminal device and return a non-negative integer representing the file descriptor.
 48140 Otherwise, -1 shall be returned and *errno* set to indicate the error.

48141 **ERRORS**

- 48142 The *posix_openpt()* function shall fail if:
- 48143 [EMFILE] All file descriptors available to the process are currently open.
 - 48144 [ENFILE] The maximum allowable number of files is currently open in the system.
- 48145 The *posix_openpt()* function may fail if:
- 48146 [EINVAL] The value of *oflag* is not valid.
 - 48147 [EAGAIN] Out of pseudo-terminal resources.
 - 48148 OB XSR [ENOSR] Out of STREAMS resources.

48149 **EXAMPLES**

48150 **Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File**
 48151 **Descriptor**

```
48152 #include <fcntl.h>
48153 #include <stdio.h>
48154 int masterfd, slavefd;
48155 char *slavedevice;
48156 masterfd = posix_openpt(O_RDWR|O_NOCTTY);
48157 if (masterfd == -1
48158     || grantpt (masterfd) == -1
```

```

48159         || unlockpt (masterfd) == -1
48160         || (slavedevice = ptsname (masterfd)) == NULL)
48161         return -1;

48162     printf("slave device is: %s\n", slavedevice);

48163     slavefd = open(slavedevice, O_RDWR|O_NOCTTY);
48164     if (slavefd < 0)
48165         return -1;

```

48166 APPLICATION USAGE

48167 This function is a method for portably obtaining a file descriptor of a master terminal device for
 48168 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and
 48169 ownership permissions, and to obtain the name of the slave device, respectively.

48170 RATIONALE

48171 The standard developers considered the matter of adding a special device for cloning master
 48172 pseudo-terminals: the */dev/ptmx* device. However, consensus could not be reached, and it was
 48173 felt that adding a new function would permit other implementations. The *posix_openpt()*
 48174 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.

48175 On implementations supporting the */dev/ptmx* clone device, opening the master device of a
 48176 pseudo-terminal is simply:

```

48177     mfdp = open("/dev/ptmx", oflag );
48178     if (mfdp < 0)
48179         return -1;

```

48180 FUTURE DIRECTIONS

48181 None.

48182 SEE ALSO

48183 [Section 2.14](#) (on page 549), *grantpt()*, *open()*, *ptsname()*, *unlockpt()*

48184 XBD [<fcntl.h>](#), [<stdlib.h>](#)

48185 CHANGE HISTORY

48186 First released in Issue 6.

48187 Issue 7

48188 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

48189 SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.

48190 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0253 [835] and XSH/TC2-2008/0254
 48191 [835] are applied.

48192 **NAME**

48193 posix_spawn, posix_spawnnp — spawn a process (**ADVANCED REALTIME**)

48194 **SYNOPSIS**

```
48195 SPN #include <spawn.h>
48196 int posix_spawn(pid_t *restrict pid, const char *restrict path,
48197               const posix_spawn_file_actions_t *file_actions,
48198               const posix_spawnattr_t *restrict attrp,
48199               char *const argv[restrict], char *const envp[restrict]);
48200 int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
48201                 const posix_spawn_file_actions_t *file_actions,
48202                 const posix_spawnattr_t *restrict attrp,
48203                 char *const argv[restrict], char *const envp[restrict]);
```

48204 **DESCRIPTION**

48205 The *posix_spawn()* and *posix_spawnnp()* functions shall create a new process (child process) from
 48206 the specified process image. The new process image shall be constructed from a regular
 48207 executable file called the new process image file.

48208 When a C program is executed as the result of this call, it shall be entered as a C-language
 48209 function call as follows:

```
48210 int main(int argc, char *argv[]);
```

48211 where *argc* is the argument count and *argv* is an array of character pointers to the arguments
 48212 themselves. In addition, the following variable:

```
48213 extern char **environ;
```

48214 shall be initialized as a pointer to an array of character pointers to the environment strings.

48215 The argument *argv* is an array of character pointers to null-terminated strings. The last member
 48216 of this array shall be a null pointer and is not counted in *argc*. These strings constitute the
 48217 argument list available to the new process image. The value in *argv*[0] should point to a filename
 48218 string that is associated with the process image being started by the *posix_spawn()* or
 48219 *posix_spawnnp()* function.

48220 The argument *envp* is an array of character pointers to null-terminated strings. These strings
 48221 constitute the environment for the new process image. The environment array is terminated by a
 48222 null pointer.

48223 The number of bytes available for the combined argument and environment lists of the child
 48224 process is {ARG_MAX}. The implementation shall specify in the system documentation (see
 48225 XBD Chapter 2, on page 15) whether any list overhead, such as length words, null terminators,
 48226 pointers, or alignment bytes, is included in this total.

48227 The *path* argument to *posix_spawn()* is a pathname that identifies the new process image file to
 48228 execute.

48229 The *file* parameter to *posix_spawnnp()* shall be used to construct a pathname that identifies the
 48230 new process image file. If the *file* parameter contains a <slash> character, the *file* parameter shall
 48231 be used as the pathname for the new process image file. Otherwise, the path prefix for this file
 48232 shall be obtained by a search of the directories passed as the environment variable *PATH* (see
 48233 XBD Chapter 8, on page 173). If this environment variable is not defined, the results of the
 48234 search are implementation-defined.

48235 If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open

48236 in the child process, except for those whose close-on-exec flag FD_CLOEXEC is set (see *fcntl()*).
 48237 For those file descriptors that remain open, the child process shall not inherit any file locks, but
 48238 all remaining attributes of the corresponding open file descriptions (see *fcntl()*), shall remain
 48239 unchanged.

48240 If *file_actions* is not NULL, then the file descriptors open in the child process shall be those open
 48241 in the calling process as modified by the spawn file actions object pointed to by *file_actions* and
 48242 the FD_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have
 48243 been processed. The effective order of processing the spawn file actions shall be:

- 48244 1. The set of open file descriptors for the child process shall initially be the same set as is
 48245 open for the calling process. The child process shall not inherit any file locks, but all
 48246 remaining attributes of the corresponding open file descriptions (see *fcntl()*), shall remain
 48247 unchanged.
- 48248 2. The signal mask, signal default actions, and the effective user and group IDs for the child
 48249 process shall be changed as specified in the attributes object referenced by *attrp*.
- 48250 3. The file actions specified by the spawn file actions object shall be performed in the order
 48251 in which they were added to the spawn file actions object.
- 48252 4. Any file descriptor that has its FD_CLOEXEC flag set (see *fcntl()*) shall be closed.

48253 If file descriptor 0, 1, or 2 would otherwise be closed in the new process image created by
 48254 *posix_spawn()* or *posix_spawnp()*, implementations may open an unspecified file for the file
 48255 descriptor in the new process image. If a standard utility or a conforming application is executed
 48256 with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the
 48257 environment in which the utility or application is executed shall be deemed non-conforming,
 48258 and consequently the utility or application might not behave as described in this standard.

48259 The **posix_spawnattr_t** spawn attributes object type is defined in **<spawn.h>**. It shall contain at
 48260 least the attributes defined below.

48261 If the POSIX_SPAWN_SETPGROUP flag is set in the *spawn_flags* attribute of the object referenced
 48262 by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process
 48263 group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

48264 As a special case, if the POSIX_SPAWN_SETPGROUP flag is set in the *spawn_flags* attribute of
 48265 the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero,
 48266 then the child shall be in a new process group with a process group ID equal to its process ID.

48267 If the POSIX_SPAWN_SETPGROUP flag is not set in the *spawn_flags* attribute of the object
 48268 referenced by *attrp*, the new child process shall inherit the parent's process group.

48269 PS If the POSIX_SPAWN_SETSCHEDPARAM flag is set in the *spawn_flags* attribute of the object
 48270 referenced by *attrp*, but POSIX_SPAWN_SETSCHEDULER is not set, the new process image
 48271 shall initially have the scheduling policy of the calling process with the scheduling parameters
 48272 specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

48273 If the POSIX_SPAWN_SETSCHEDULER flag is set in the *spawn_flags* attribute of the object
 48274 referenced by *attrp* (regardless of the setting of the POSIX_SPAWN_SETSCHEDPARAM flag),
 48275 the new process image shall initially have the scheduling policy specified in the *spawn-*
 48276 *schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in
 48277 the *spawn-schedparam* attribute of the same object.

48278 The POSIX_SPAWN_RESETPID flag in the *spawn_flags* attribute of the object referenced by *attrp*
 48279 governs the effective user ID of the child process. If this flag is not set, the child process shall
 48280 inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the

48281 child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit
48282 of the new process image file is set, the effective user ID of the child process shall become that
48283 file's owner ID before the new process image begins execution.

48284 The POSIX_SPAWN_RESETEUIDS flag in the *spawn_flags* attribute of the object referenced by *attrp*
48285 also governs the effective group ID of the child process. If this flag is not set, the child process
48286 shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID
48287 of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID
48288 mode bit of the new process image file is set, the effective group ID of the child process shall
48289 become that file's group ID before the new process image begins execution.

48290 If the POSIX_SPAWN_SETSIGMASK flag is set in the *spawn_flags* attribute of the object
48291 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*
48292 *sigmask* attribute of the object referenced by *attrp*.

48293 If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn_flags* attribute of the object referenced
48294 by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to
48295 their default actions in the child process. Signals set to the default action in the parent process
48296 shall be set to the default action in the child process.

48297 Signals set to be caught by the calling process shall be set to the default action in the child
48298 process.

48299 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be
48300 ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag
48301 being set in the *spawn_flags* attribute of the object referenced by *attrp* and the signals being
48302 indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

48303 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the
48304 SIGCHLD signal is set to be ignored or to the default action in the child process, unless
48305 otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags*
48306 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the
48307 *spawn-sigdefault* attribute of the object referenced by *attrp*.

48308 If the value of the *attrp* pointer is NULL, then the default values are used.

48309 All process attributes, other than those influenced by the attributes set in the object referenced
48310 by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall
48311 appear in the new process image as though *fork()* had been called to create a child process and
48312 then a member of the *exec* family of functions had been called by the child process to execute the
48313 new process image.

48314 It is implementation-defined whether the fork handlers are run when *posix_spawn()* or
48315 *posix_spawnnp()* is called.

48316 RETURN VALUE

48317 Upon successful completion, *posix_spawn()* and *posix_spawnnp()* shall return the process ID of the
48318 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and
48319 shall return zero as the function return value. Otherwise, no child process shall be created, the
48320 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number
48321 shall be returned as the function return value to indicate the error. If the *pid* argument is a null
48322 pointer, the process ID of the child is not returned to the caller.

48323 ERRORS

48324 These functions may fail if:

48325 [EINVAL] The value specified by *file_actions* or *attrp* is invalid.

48326 If this error occurs after the calling process successfully returns from the *posix_spawn()* or
48327 *posix_spawnnp()* function, the child process may exit with exit status 127.

48328 If *posix_spawn()* or *posix_spawnnp()* fail for any of the reasons that would cause *fork()* or one of
48329 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and
48330 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child
48331 process shall exit with exit status 127).

48332 If POSIX_SPAWN_SETPGROUP is set in the *spawn_flags* attribute of the object referenced by
48333 *attrp*, and *posix_spawn()* or *posix_spawnnp()* fails while changing the child's process group, an
48334 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling
48335 process successfully returns, the child process shall exit with exit status 127).

48336 PS If POSIX_SPAWN_SETSCHEDPARAM is set and POSIX_SPAWN_SETSCHEDULER is not set in
48337 the *spawn_flags* attribute of the object referenced by *attrp*, then if *posix_spawn()* or *posix_spawnnp()*
48338 fails for any of the reasons that would cause *sched_setparam()* to fail, an error value shall be
48339 returned as described by *sched_setparam()* (or, if the error occurs after the calling process
48340 successfully returns, the child process shall exit with exit status 127).

48341 If POSIX_SPAWN_SETSCHEDULER is set in the *spawn_flags* attribute of the object referenced by
48342 *attrp*, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause
48343 *sched_setscheduler()* to fail, an error value shall be returned as described by *sched_setscheduler()*
48344 (or, if the error occurs after the calling process successfully returns, the child process shall exit
48345 with exit status 127).

48346 If the *file_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be
48347 performed, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause
48348 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*, *dup2()*,
48349 and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the
48350 child process shall exit with exit status 127). An open file action may, by itself, result in any of
48351 the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

EXAMPLES

48352 None.

APPLICATION USAGE

48353 These functions are part of the Spawn option and need not be provided on all implementations.

48354 See also the APPLICATION USAGE section for *exec*.

RATIONALE

48358 The *posix_spawn()* function and its close relation *posix_spawnnp()* have been introduced to
48359 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or
48360 impossible to implement without swapping or dynamic address translation.

48361 Swapping is generally too slow for a realtime environment.

48362 Dynamic address translation is not available everywhere that POSIX might be useful.

48363 Processes are too useful to simply option out of POSIX whenever it must run without
48364 address translation or other MMU services.

48365 Thus, POSIX needs process creation and file execution primitives that can be efficiently
48366 implemented without address translation or other MMU services.

48367 The *posix_spawn()* function is implementable as a library routine, but both *posix_spawn()* and
48368 *posix_spawnnp()* are designed as kernel operations. Also, although they may be an efficient

48369 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation
48370 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for
48371 *fork()/exec*.

48372 This view of the role of *posix_spawn()* and *posix_spawnnp()* influenced the design of their API. It
48373 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified
48374 operations of any sort are permitted between the creation of the child process and the execution
48375 of the new process image; any attempt to reach that level would need to provide a programming
48376 language as parameters. Instead, *posix_spawn()* and *posix_spawnnp()* are process creation
48377 primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package
48378 *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX
48379 systems, but with some POSIX-specific additions.

48380 To achieve its coverage goals, *posix_spawn()* and *posix_spawnnp()* have control of six types of
48381 inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and
48382 whether each signal ignored in the parent will remain ignored in the child, or be reset to its
48383 default action in the child.

48384 Control of file descriptors is required to allow an independently written child process image to
48385 access data streams opened by and even generated or read by the parent process without being
48386 specifically coded to know which parent files and file descriptors are to be used. Control of the
48387 process group ID is required to control how the job control of the child process relates to that of
48388 the parent.

48389 Control of the signal mask and signal defaulting is sufficient to support the implementation of
48390 *system()*. Although support for *system()* is not explicitly one of the goals for *posix_spawn()* and
48391 *posix_spawnnp()*, it is covered under the “at least 50%” coverage goal.

48392 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of
48393 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*
48394 family of functions should fully specify open file inheritance. The implementation need make no
48395 decisions regarding the set of open file descriptors when the child process image begins
48396 execution, those decisions having already been made by the caller and expressed as the set of
48397 open file descriptors and their *FD_CLOEXEC* flags at the time of the call and the spawn file
48398 actions object specified in the call. We have been assured that in cases where the POSIX
48399 *Start_Process* Ada primitives have been implemented in a library, this method of controlling file
48400 descriptor inheritance may be implemented very easily.

48401 We can identify several problems with *posix_spawn()* and *posix_spawnnp()*, but there does not
48402 appear to be a solution that introduces fewer problems. Environment modification for child
48403 process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the
48404 parent process, and since the parent generally wants to save its context, it is more costly than
48405 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a
48406 multi-threaded process temporarily, since all threads must agree when it is safe for the
48407 environment to be changed. However, this cost is only borne by those invocations of
48408 *posix_spawn()* and *posix_spawnnp()* that use the additional functionality. Since extensive
48409 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping
48410 much of the environment control out of *posix_spawn()* and *posix_spawnnp()* is appropriate design.

48411 The *posix_spawn()* and *posix_spawnnp()* functions do not have all the power of *fork()/exec*. This is
48412 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to
48413 duplicate its functionality in a simple, fast function with no special hardware requirements. It is
48414 worth noting that *posix_spawn()* and *posix_spawnnp()* are very similar to the process creation
48415 operations on many operating systems that are not UNIX systems.

48416 **Requirements**

48417 The requirements for *posix_spawn()* and *posix_spawnp()* are:

48418 They must be implementable without an MMU or unusual hardware.

48419 They must be compatible with existing POSIX standards.

48420 Additional goals are:

48421 They should be efficiently implementable.

48422 They should be able to replace at least 50% of typical executions of *fork()*.

48423 A system with *posix_spawn()* and *posix_spawnp()* and without *fork()* should be useful, at
48424 least for realtime applications.

48425 A system with *fork()* and the *exec* family should be able to implement *posix_spawn()* and
48426 *posix_spawnp()* as library routines.

48427 **Two-Syntax**

48428 POSIX *exec* has several calling sequences with approximately the same functionality. These
48429 appear to be required for compatibility with existing practice. Since the existing practice for the
48430 *posix_spawn*()* functions is otherwise substantially unlike POSIX, we feel that simplicity
48431 outweighs compatibility. There are, therefore, only two names for the *posix_spawn*()* functions.

48432 The parameter list does not differ between *posix_spawn()* and *posix_spawnp()*; *posix_spawnp()*
48433 interprets the second parameter more elaborately than *posix_spawn()*.

48434 **Compatibility with POSIX.5 (Ada)**

48435 The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives*
48436 package from the Ada language binding to POSIX.1 encapsulate *fork()* and *exec* functionality in a
48437 manner similar to that of *posix_spawn()* and *posix_spawnp()*. Originally, in keeping with our
48438 simplicity goal, the standard developers had limited the capabilities of *posix_spawn()* and
48439 *posix_spawnp()* to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain
48440 non-default capabilities were not supported. However, based on suggestions by the ballot group
48441 to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings
48442 working group member, the standard developers decided that *posix_spawn()* and *posix_spawnp()*
48443 should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The
48444 rationale is that if the Ada language binding to such a primitive had already been approved as
48445 an IEEE standard, there can be little justification for not approving the functionally-equivalent
48446 parts of a C binding. The only three capabilities provided by *posix_spawn()* and *posix_spawnp()*
48447 that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the
48448 child's process group ID, the set of signals to be reset to default signal handling in the child
48449 process, and the child's scheduling policy and parameters.

48450 For the Ada language binding for *Start_Process* to be implemented with *posix_spawn()*, that
48451 binding would need to explicitly pass an empty signal mask and the parent's environment to
48452 *posix_spawn()* whenever the caller of *Start_Process* allowed these arguments to default, since
48453 *posix_spawn()* does not provide such defaults. The ability of *Start_Process* to mask user-specified
48454 signals during its execution is functionally unique to the Ada language binding and must be
48455 dealt with in the binding separately from the call to *posix_spawn()*.

48456 **Process Group**

48457 The process group inheritance field can be used to join the child process with an existing process
48458 group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*,
48459 the *setpgid()* mechanism will place the child process in a new process group.

48460 **Threads**

48461 Without the *posix_spawn()* and *posix_spawnp()* functions, systems without address translation
48462 can still use threads to give an abstraction of concurrency. In many cases, thread creation
48463 suffices, but it is not always a good substitute. The *posix_spawn()* and *posix_spawnp()* functions
48464 are considerably “heavier” than thread creation. Processes have several important attributes that
48465 threads do not. Even without address translation, a process may have base-and-bound memory
48466 protection. Each process has a process environment including security attributes and file
48467 capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-
48468 uniform-memory-architecture multi-processors better than threads, and they are more
48469 convenient to use for activities that are not closely linked.

48470 The *posix_spawn()* and *posix_spawnp()* functions may not bring support for multiple processes to
48471 every configuration. Process creation is not the only piece of operating system support required
48472 to support multiple processes. The total cost of support for multiple processes may be quite high
48473 in some circumstances. Existing practice shows that support for multiple processes is
48474 uncommon and threads are common among “tiny kernels”. There should, therefore, probably
48475 continue to be AEPs for operating systems with only one process.

48476 **Asynchronous Error Notification**

48477 A library implementation of *posix_spawn()* or *posix_spawnp()* may not be able to detect all
48478 possible errors before it forks the child process. POSIX.1-2017 provides for an error indication
48479 returned from a child process which could not successfully complete the spawn operation via a
48480 special exit status which may be detected using the status value returned by *wait()*, *waitid()*, and
48481 *waitpid()*.

48482 The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of
48483 returning API errors, but they are the only path available to a library implementation. Thus, an
48484 implementation may cause the child process to exit with exit status 127 for any error detected
48485 during the spawn process after the *posix_spawn()* or *posix_spawnp()* function has successfully
48486 returned.

48487 The standard developers had proposed using two additional macros to interpret *stat_val*. The
48488 first, *WIFSPAWNFAIL*, would have detected a status that indicated that the child exited because
48489 of an error detected during the *posix_spawn()* or *posix_spawnp()* operations rather than during
48490 actual execution of the child process image; the second, *WSPAWNERRNO*, would have
48491 extracted the error value if *WIFSPAWNFAIL* indicated a failure. Unfortunately, the ballot group
48492 strongly opposed this because it would make a library implementation of *posix_spawn()* or
48493 *posix_spawnp()* dependent on kernel modifications to *waitpid()* to be able to embed special
48494 information in *stat_val* to indicate a spawn failure.

48495 The 8 bits of child process exit status that are guaranteed by POSIX.1-2017 to be accessible to the
48496 waiting parent process are insufficient to disambiguate a spawn error from any other kind of
48497 error that may be returned by an arbitrary process image. No other bits of the exit status are
48498 required to be visible in *stat_val*, so these macros could not be strictly implemented at the library
48499 level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this
48500 value by *system()* and *popen()* to signal failures in these operations that occur after the function
48501 has returned but before a shell is able to execute. The exit status of 127 does not uniquely

48502 identify this class of error, nor does it provide any detailed information on the nature of the
 48503 failure. Note that a kernel implementation of *posix_spawn()* or *posix_spawnp()* is permitted (and
 48504 encouraged) to return any possible error as the function value, thus providing more detailed
 48505 failure information to the parent process.

48506 Thus, no special macros are available to isolate asynchronous *posix_spawn()* or *posix_spawnp()*
 48507 errors. Instead, errors detected by the *posix_spawn()* or *posix_spawnp()* operations in the context
 48508 of the child process before the new process image executes are reported by setting the child's exit
 48509 status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the
 48510 *stat_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that
 48511 other status values with which the child process image may exit (before the parent can
 48512 conclusively determine that the child process image has begun execution) are distinct from exit
 48513 status 127.

48514 FUTURE DIRECTIONS

48515 None.

48516 SEE ALSO

48517 *alarm()*, *chmod()*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *fstatat()*, *kill()*, *open()*,
 48518 *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*,
 48519 *posix_spawn_file_actions_destroy()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
 48520 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
 48521 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *sched_setparam()*,
 48522 *sched_setscheduler()*, *setpgid()*, *setuid()*, *times()*, *wait()*, *waitid()*

48523 XBD Chapter 8 (on page 173), [<spawn.h>](#)

48524 CHANGE HISTORY

48525 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48526 IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are
 48527 changed as well as the signal mask in step 2.

48528 IEEE PASC Interpretation 1003.1 #132 is applied.

48529 Issue 7

48530 Functionality relating to the Threads option is moved to the Base.

48531 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0433 [291], XSH/TC1-2008/0434 [173],
 48532 and XSH/TC1-2008/0435 [173] are applied.

48533 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0255 [824] is applied.

48534 **NAME**

48535 posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen ‡' add close or open
48536 action to spawn file actions object (**ADVANCED REALTIME**)

48537 **SYNOPSIS**

```
48538 SPN #include <spawn.h>
48539 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
48540     *file_actions, int fildes);
48541 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
48542     *restrict file_actions, int fildes,
48543     const char *restrict path, int oflag, mode_t mode);
```

48544 **DESCRIPTION**

48545 These functions shall add or delete a close or open action to a spawn file actions object.

48546 A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in **<spawn.h>**) and is
48547 used to specify a series of actions to be performed by a *posix_spawn()* or *posix_spawnp()*
48548 operation in order to arrive at the set of open file descriptors for the child process given the set
48549 of open file descriptors of the parent. POSIX.1-2017 does not define comparison or assignment
48550 operators for the type **posix_spawn_file_actions_t**.

48551 A spawn file actions object, when passed to *posix_spawn()* or *posix_spawnp()*, shall specify how
48552 the set of open file descriptors in the calling process is transformed into a set of potentially open
48553 file descriptors for the spawned process. This transformation shall be as if the specified sequence
48554 of actions was performed exactly once, in the context of the spawned process (prior to execution
48555 of the new process image), in the order in which the actions were added to the object;
48556 additionally, when the new process image is executed, any file descriptor (from this new set)
48557 which has its FD_CLOEXEC flag set shall be closed (see *posix_spawn()*).

48558 The *posix_spawn_file_actions_addclose()* function shall add a *close* action to the object referenced
48559 by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if *close(fildes)* had been
48560 called) when a new process is spawned using this file actions object.

48561 The *posix_spawn_file_actions_addopen()* function shall add an *open* action to the object referenced
48562 by *file_actions* that shall cause the file named by *path* to be opened (as if *open(path, oflag, mode)*
48563 had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a
48564 new process is spawned using this file actions object. If *fildes* was already an open file descriptor,
48565 it shall be closed before the new file is opened.

48566 The string described by *path* shall be copied by the *posix_spawn_file_actions_addopen()* function.

48567 **RETURN VALUE**

48568 Upon successful completion, these functions shall return zero; otherwise, an error number shall
48569 be returned to indicate the error.

48570 **ERRORS**

48571 The *posix_spawn_file_actions_addopen()* function shall fail if:

48572 [EBADF] The value specified by *fildes* is negative or greater than or equal to
48573 {OPEN_MAX}.

48574 The *posix_spawn_file_actions_addclose()* function shall fail if:

48575 [EBADF] The value specified by *fildes* is negative.

48576 These functions may fail if:

48577 [EINVAL] The value specified by *file_actions* is invalid.

48578 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

48579 It shall not be considered an error for the *fildevs* argument passed to these functions to specify a
 48580 file descriptor for which the specified operation could not be performed at the time of the call.
 48581 Any such error will be detected when the associated file actions object is later used during a
 48582 *posix_spawn()* or *posix_spawnnp()* operation.

48583 EXAMPLES

48584 None.

48585 APPLICATION USAGE

48586 These functions are part of the Spawn option and need not be provided on all implementations.

48587 Implementations may use file descriptors that must be inherited into child processes for the
 48588 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,
 48589 an application that calls *posix_spawn_file_actions_addclose()* with an arbitrary integer risks non-
 48590 conforming behavior, and this function can only portably be used to close file descriptor values
 48591 that the application has obtained through explicit actions, or for the three file descriptors
 48592 corresponding to the standard file streams. In order to avoid a race condition of leaking an
 48593 unintended file descriptor into a child process, an application should consider opening all file
 48594 descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited
 48595 across *exec*.

48596 RATIONALE

48597 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,
 48598 and *open()* operations to be used by *posix_spawn()* or *posix_spawnnp()* to arrive at the set of open
 48599 file descriptors inherited by the spawned process from the set of open file descriptors in the
 48600 parent at the time of the *posix_spawn()* or *posix_spawnnp()* call. It had been suggested that the
 48601 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files
 48602 which need to be opened for use by the spawned process can be handled either by having the
 48603 calling process open them before the *posix_spawn()* or *posix_spawnnp()* call (and close them after),
 48604 or by passing pathnames to the spawned process (in *argv*) so that it may open them itself. The
 48605 standard developers recommend that applications use one of these two methods when practical,
 48606 since detailed error status on a failed open operation is always available to the application this
 48607 way. However, the standard developers feel that allowing a spawn file actions object to specify
 48608 open operations is still appropriate because:

- 48609 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 48610 2. It supports the I/O redirection paradigm commonly employed by POSIX programs
 48611 designed to be invoked from a shell. When such a program is the child process, it may not
 48612 be designed to open files on its own.
- 48613 3. It allows file opens that might otherwise fail or violate file ownership/access rights if
 48614 executed by the parent process.

48615 Regarding 2. above, note that the spawn open file action provides to *posix_spawn()* and
 48616 *posix_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only
 48617 without the intervening execution of a shell; for example:

```
48618 system ("myprog <file1 3<file2");
```

48619 Regarding 3. above, note that if the calling process needs to open one or more files for access by
 48620 the spawned process, but has insufficient spare file descriptors, then the open action is necessary

48621 to allow the *open()* to occur in the context of the child process after other file descriptors have
48622 been closed (that must remain open in the parent).

48623 Additionally, if a parent is executed from a file having a “set-user-id” mode bit set and the
48624 POSIX_SPAWN_RESETEUIDS flag is set in the spawn attributes, a file created within the parent
48625 process will (possibly incorrectly) have the parent’s effective user ID as its owner, whereas a file
48626 created via an *open()* action during *posix_spawn()* or *posix_spawnnp()* will have the parent’s real
48627 ID as its owner; and an open by the parent process may successfully open a file to which the real
48628 user should not have access or fail to open a file to which the real user should have access.

48629 File Descriptor Mapping

48630 The standard developers had originally proposed using an array which specified the mapping of
48631 child file descriptors back to those of the parent. It was pointed out by the ballot group that it is
48632 not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn()*
48633 or *posix_spawnnp()* without provision for one or more spare file descriptor entries (which simply
48634 may not be available). Such an array requires that an implementation develop a complex
48635 strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor
48636 at the wrong time.

48637 It was noted by a member of the Ada Language Bindings working group that the approved Ada
48638 Language *Start_Process* family of POSIX process primitives use a caller-specified set of file
48639 actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very
48640 flexible way, yet no such problems exist because the burden of determining how to achieve the
48641 final file descriptor mapping is completely on the application. Furthermore, although the file
48642 actions interface appears frightening at first glance, it is actually quite simple to implement in
48643 either a library or the kernel.

48644 The *posix_spawn_file_actions_addclose()* function is not required to check whether the file
48645 descriptor is less than {OPEN_MAX} because on some implementations {OPEN_MAX} reflects
48646 the RLIMIT_NOFILE soft limit and therefore calling *setrlimit()* to reduce this limit can result in
48647 an {OPEN_MAX} value less than or equal to an already open file descriptor. Applications need
48648 to be able to close such file descriptors on spawn. On implementations where {OPEN_MAX}
48649 does not change, it is recommended that *posix_spawn_file_actions_addclose()* should return
48650 [EBADF] if *fd* is greater than or equal to {OPEN_MAX}.

48651 FUTURE DIRECTIONS

48652 None.

48653 SEE ALSO

48654 *close()*, *dup()*, *open()*, *posix_spawn()*, *posix_spawn_file_actions_adddup2()*,
48655 *posix_spawn_file_actions_destroy()*

48656 XBD <spawn.h>

48657 CHANGE HISTORY

48658 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48659 IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the
48660 string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen()* function.

48661 Issue 7

48662 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0436 [418], XSH/TC1-2008/0437 [149],
48663 XSH/TC1-2008/0438 [291], and XSH/TC1-2008/0439 [418] are applied.

48664 **NAME**

48665 posix_spawn_file_actions_adddup2 ‡' add dup2 action to spawn file actions object
48666 (ADVANCED REALTIME)

48667 **SYNOPSIS**

```
48668 SPN    #include <spawn.h>
48669
48669        int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t
48670            *file_actions, int fildes, int newfildes);
```

48671 **DESCRIPTION**

48672 The *posix_spawn_file_actions_adddup2()* function shall add a *dup2()* action to the object
48673 referenced by *file_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as
48674 if *dup2(fildes, newfildes)* had been called) when a new process is spawned using this file actions
48675 object.

48676 A spawn file actions object is as defined in *posix_spawn_file_actions_addclose()*.

48677 **RETURN VALUE**

48678 Upon successful completion, the *posix_spawn_file_actions_adddup2()* function shall return zero;
48679 otherwise, an error number shall be returned to indicate the error.

48680 **ERRORS**

48681 The *posix_spawn_file_actions_adddup2()* function shall fail if:

48682 [EBADF] The value specified by *fildes* or *newfildes* is negative or greater than or equal to
48683 {OPEN_MAX}.

48684 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

48685 The *posix_spawn_file_actions_adddup2()* function may fail if:

48686 [EINVAL] The value specified by *file_actions* is invalid.

48687 It shall not be considered an error for the *fildes* argument passed to the
48688 *posix_spawn_file_actions_adddup2()* function to specify a file descriptor for which the specified
48689 operation could not be performed at the time of the call. Any such error will be detected when
48690 the associated file actions object is later used during a *posix_spawn()* or *posix_spawnnp()*
48691 operation.

48692 **EXAMPLES**

48693 None.

48694 **APPLICATION USAGE**

48695 The *posix_spawn_file_actions_adddup2()* function is part of the Spawn option and need not be
48696 provided on all implementations.

48697 Implementations may use file descriptors that must be inherited into child processes for the
48698 child process to remain conforming, such as for message catalog or tracing purposes. Therefore,
48699 an application that calls *posix_spawn_file_actions_adddup2()* with an arbitrary integer for *newfildes*
48700 risks non-conforming behavior, and this function can only portably be used to overwrite file
48701 descriptor values that the application has obtained through explicit actions, or for the three file
48702 descriptors corresponding to the standard file streams. In order to avoid a race condition of
48703 leaking an unintended file descriptor into a child process, an application should consider
48704 opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended
48705 to be inherited across *exec*.

48706 RATIONALE

48707 Refer to the RATIONALE section in *posix_spawn_file_actions_addclose()*.

48708 FUTURE DIRECTIONS

48709 None.

48710 SEE ALSO

48711 *dup()*, *posix_spawn()*, *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_destroy()*

48712 XBD <**spawn.h**>

48713 CHANGE HISTORY

48714 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48715 IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the

48716 *newfiles* argument in addition to *files*.

48717 Issue 7

48718 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0440 [149] is applied.

48719 **NAME**

48720 posix_spawn_file_actions_addopen ‡' add open action to spawn file actions object
48721 (**ADVANCED REALTIME**)

48722 **SYNOPSIS**

```
48723 SPN #include <spawn.h>  
48724 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t  
48725 *restrict file_actions, int fildes,  
48726 const char *restrict path, int oflag, mode_t mode);
```

48727 **DESCRIPTION**

48728 Refer to [posix_spawn_file_actions_addclose\(\)](#).

48729 **NAME**

48730 posix_spawn_file_actions_destroy, posix_spawn_file_actions_init — destroy and initialize
 48731 spawn file actions object (**ADVANCED REALTIME**)

48732 **SYNOPSIS**

```
48733 SPN #include <spawn.h>
48734 int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
48735     *file_actions);
48736 int posix_spawn_file_actions_init(posix_spawn_file_actions_t
48737     *file_actions);
```

48738 **DESCRIPTION**

48739 The *posix_spawn_file_actions_destroy()* function shall destroy the object referenced by *file_actions*;
 48740 the object becomes, in effect, uninitialized. An implementation may cause
 48741 *posix_spawn_file_actions_destroy()* to set the object referenced by *file_actions* to an invalid value. A
 48742 destroyed spawn file actions object can be reinitialized using *posix_spawn_file_actions_init()*; the
 48743 results of otherwise referencing the object after it has been destroyed are undefined.

48744 The *posix_spawn_file_actions_init()* function shall initialize the object referenced by *file_actions* to
 48745 contain no file actions for *posix_spawn()* or *posix_spawnnp()* to perform.

48746 A spawn file actions object is as defined in *posix_spawn_file_actions_addclose()*.

48747 The effect of initializing an already initialized spawn file actions object is undefined.

48748 **RETURN VALUE**

48749 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 48750 be returned to indicate the error.

48751 **ERRORS**

48752 The *posix_spawn_file_actions_init()* function shall fail if:

48753 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

48754 The *posix_spawn_file_actions_destroy()* function may fail if:

48755 [EINVAL] The value specified by *file_actions* is invalid.

48756 **EXAMPLES**

48757 None.

48758 **APPLICATION USAGE**

48759 These functions are part of the Spawn option and need not be provided on all implementations.

48760 **RATIONALE**

48761 Refer to the RATIONALE section in *posix_spawn_file_actions_addclose()*.

48762 **FUTURE DIRECTIONS**

48763 None.

48764 **SEE ALSO**

48765 *posix_spawn()*, *posix_spawn_file_actions_addclose()*

48766 XBD <spawn.h>

48767 **CHANGE HISTORY**

48768 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48769 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

48770 **NAME**

48771 posix_spawnattr_destroy, posix_spawnattr_init ‡destroy and initialize spawn attributes object
48772 (ADVANCED REALTIME)

48773 **SYNOPSIS**

```
48774 SPN #include <spawn.h>
48775 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
48776 int posix_spawnattr_init(posix_spawnattr_t *attr);
```

48777 **DESCRIPTION**

48778 The *posix_spawnattr_destroy()* function shall destroy a spawn attributes object. A destroyed *attr*
48779 attributes object can be reinitialized using *posix_spawnattr_init()*; the results of otherwise
48780 referencing the object after it has been destroyed are undefined. An implementation may cause
48781 *posix_spawnattr_destroy()* to set the object referenced by *attr* to an invalid value.

48782 The *posix_spawnattr_init()* function shall initialize a spawn attributes object *attr* with the default
48783 value for all of the individual attributes used by the implementation. Results are undefined if
48784 *posix_spawnattr_init()* is called specifying an already initialized *attr* attributes object.

48785 A spawn attributes object is of type **posix_spawnattr_t** (defined in **<spawn.h>**) and is used to
48786 specify the inheritance of process attributes across a spawn operation. POSIX.1-2017 does not
48787 define comparison or assignment operators for the type **posix_spawnattr_t**.

48788 Each implementation shall document the individual attributes it uses and their default values
48789 unless these values are defined by POSIX.1-2017. Attributes not defined by POSIX.1-2017, their
48790 default values, and the names of the associated functions to get and set those attribute values are
48791 implementation-defined.

48792 The resulting spawn attributes object (possibly modified by setting individual attribute values),
48793 is used to modify the behavior of *posix_spawn()* or *posix_spawnnp()*. After a spawn attributes
48794 object has been used to spawn a process by a call to a *posix_spawn()* or *posix_spawnnp()*, any
48795 function affecting the attributes object (including destruction) shall not affect any process that
48796 has been spawned in this way.

48797 **RETURN VALUE**

48798 Upon successful completion, *posix_spawnattr_destroy()* and *posix_spawnattr_init()* shall return
48799 zero; otherwise, an error number shall be returned to indicate the error.

48800 **ERRORS**

48801 The *posix_spawnattr_init()* function shall fail if:

48802 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

48803 The *posix_spawnattr_destroy()* function may fail if:

48804 [EINVAL] The value specified by *attr* is invalid.

48805 **EXAMPLES**

48806 None.

48807 **APPLICATION USAGE**

48808 These functions are part of the Spawn option and need not be provided on all implementations.

48809 **RATIONALE**

48810 The original spawn interface proposed in POSIX.1-2017 defined the attributes that specify the
48811 inheritance of process attributes across a spawn operation as a structure. In order to be able to
48812 separate optional individual attributes under their appropriate options (that is, the *spawn-*
48813 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and

48814 also for extensibility and consistency with the newer POSIX interfaces, the attributes interface
48815 has been changed to an opaque data type. This interface now consists of the type
48816 **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to
48817 initialize or destroy the attributes object, and to set or get each individual attribute. Although the
48818 new object-oriented interface is more verbose than the original structure, it is simple to use,
48819 more extensible, and easy to implement.

48820 FUTURE DIRECTIONS

48821 None.

48822 SEE ALSO

48823 *posix_spawn()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
48824 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
48825 *posix_spawnattr_getsigmask()*

48826 XBD <spawn.h>

48827 CHANGE HISTORY

48828 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48829 IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already
48830 initialized spawn attributes option is undefined.

48831 **NAME**

48832 posix_spawnattr_getflags, posix_spawnattr_setflags ‡get and set the spawn-flags attribute of a
 48833 spawn attributes object (**ADVANCED REALTIME**)

48834 **SYNOPSIS**

```
48835 SPN #include <spawn.h>
48836 int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
48837     short *restrict flags);
48838 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

48839 **DESCRIPTION**

48840 The *posix_spawnattr_getflags()* function shall obtain the value of the *spawn-flags* attribute from the
 48841 attributes object referenced by *attr*.

48842 The *posix_spawnattr_setflags()* function shall set the *spawn-flags* attribute in an initialized
 48843 attributes object referenced by *attr*.

48844 The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the
 48845 new process image when invoking *posix_spawn()* or *posix_spawnnp()*. It is the bitwise-inclusive
 48846 OR of zero or more of the following flags:

- 48847 POSIX_SPAWN_RESETIDS
- 48848 POSIX_SPAWN_SETPGROUP
- 48849 POSIX_SPAWN_SETSIGDEF
- 48850 POSIX_SPAWN_SETSIGMASK
- 48851 PS POSIX_SPAWN_SETSCHEDPARAM
- 48852 POSIX_SPAWN_SETSCHEDULER

48853 These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags
 48854 were set.

48855 **RETURN VALUE**

48856 Upon successful completion, *posix_spawnattr_getflags()* shall return zero and store the value of
 48857 the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an
 48858 error number shall be returned to indicate the error.

48859 Upon successful completion, *posix_spawnattr_setflags()* shall return zero; otherwise, an error
 48860 number shall be returned to indicate the error.

48861 **ERRORS**

48862 These functions may fail if:

- 48863 [EINVAL] The value specified by *attr* is invalid.
- 48864 The *posix_spawnattr_setflags()* function may fail if:
- 48865 [EINVAL] The value of the attribute being set is not valid.

48866 **EXAMPLES**

48867 None.

48868 **APPLICATION USAGE**

48869 These functions are part of the Spawn option and need not be provided on all implementations.

48870 **RATIONALE**

48871 None.

48872 **FUTURE DIRECTIONS**

48873 None.

48874 **SEE ALSO**48875 *posix_spawn(), posix_spawnattr_destroy(), posix_spawnattr_getsigdefault(),*
48876 *posix_spawnattr_getpgroup(), posix_spawnattr_getschedparam(), posix_spawnattr_getschedpolicy(),*
48877 *posix_spawnattr_getsigmask()*

48878 XBD <spawn.h>

48879 **CHANGE HISTORY**

48880 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48881 **NAME**

48882 posix_spawnattr_getpgroup, posix_spawnattr_setpgroup ‡ get and set the spawn-pgroup
48883 attribute of a spawn attributes object (ADVANCED REALTIME)

48884 **SYNOPSIS**

```
48885 SPN #include <spawn.h>
48886 int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
48887 pid_t *restrict pgroup);
48888 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

48889 **DESCRIPTION**

48890 The *posix_spawnattr_getpgroup()* function shall obtain the value of the *spawn-pgroup* attribute
48891 from the attributes object referenced by *attr*.

48892 The *posix_spawnattr_setpgroup()* function shall set the *spawn-pgroup* attribute in an initialized
48893 attributes object referenced by *attr*.

48894 The *spawn-pgroup* attribute represents the process group to be joined by the new process image
48895 in a spawn operation (if POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute). The
48896 default value of this attribute shall be zero.

48897 **RETURN VALUE**

48898 Upon successful completion, *posix_spawnattr_getpgroup()* shall return zero and store the value of
48899 the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise,
48900 an error number shall be returned to indicate the error.

48901 Upon successful completion, *posix_spawnattr_setpgroup()* shall return zero; otherwise, an error
48902 number shall be returned to indicate the error.

48903 **ERRORS**

48904 These functions may fail if:

48905 [EINVAL] The value specified by *attr* is invalid.

48906 The *posix_spawnattr_setpgroup()* function may fail if:

48907 [EINVAL] The value of the attribute being set is not valid.

48908 **EXAMPLES**

48909 None.

48910 **APPLICATION USAGE**

48911 These functions are part of the Spawn option and need not be provided on all implementations.

48912 **RATIONALE**

48913 None.

48914 **FUTURE DIRECTIONS**

48915 None.

48916 **SEE ALSO**

48917 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
48918 *posix_spawnattr_getflags()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
48919 *posix_spawnattr_getsigmask()*

48920 XBD [<spawn.h>](#)

48921 **CHANGE HISTORY**

48922 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48923 **NAME**

48924 posix_spawnattr_getschedparam, posix_spawnattr_setschedparam ‡ get and set the spawn-
48925 schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

48926 **SYNOPSIS**

```
48927 SPN PS #include <spawn.h>
48928 #include <sched.h>
48929
48929 int posix_spawnattr_getschedparam(const posix_spawnattr_t
48930     *restrict attr, struct sched_param *restrict schedparam);
48931 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
48932     const struct sched_param *restrict schedparam);
```

48933 **DESCRIPTION**

48934 The *posix_spawnattr_getschedparam()* function shall obtain the value of the *spawn-schedparam*
48935 attribute from the attributes object referenced by *attr*.

48936 The *posix_spawnattr_setschedparam()* function shall set the *spawn-schedparam* attribute in an
48937 initialized attributes object referenced by *attr*.

48938 The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new
48939 process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER or
48940 POSIX_SPAWN_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this
48941 attribute is unspecified.

48942 **RETURN VALUE**

48943 Upon successful completion, *posix_spawnattr_getschedparam()* shall return zero and store the
48944 value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam*
48945 parameter; otherwise, an error number shall be returned to indicate the error.

48946 Upon successful completion, *posix_spawnattr_setschedparam()* shall return zero; otherwise, an
48947 error number shall be returned to indicate the error.

48948 **ERRORS**

48949 These functions may fail if:

48950 [EINVAL] The value specified by *attr* is invalid.

48951 The *posix_spawnattr_setschedparam()* function may fail if:

48952 [EINVAL] The value of the attribute being set is not valid.

48953 **EXAMPLES**

48954 None.

48955 **APPLICATION USAGE**

48956 These functions are part of the Spawn and Process Scheduling options and need not be provided
48957 on all implementations.

48958 **RATIONALE**

48959 None.

48960 **FUTURE DIRECTIONS**

48961 None.

48962 **SEE ALSO**

48963 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
48964 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedpolicy()*,
48965 *posix_spawnattr_getsigmask()*

48966 XBD <sched.h>, <spawn.h>

48967 **CHANGE HISTORY**

48968 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

48969 **NAME**

48970 posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy ‡ get and set the spawn-
48971 schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

48972 **SYNOPSIS**

```
48973 SPN PS #include <spawn.h>
48974 #include <sched.h>
48975
48976 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
48977 *restrict attr, int *restrict schedpolicy);
48978 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
48979 int schedpolicy);
```

48979 **DESCRIPTION**

48980 The *posix_spawnattr_getschedpolicy()* function shall obtain the value of the *spawn-schedpolicy*
48981 attribute from the attributes object referenced by *attr*.

48982 The *posix_spawnattr_setschedpolicy()* function shall set the *spawn-schedpolicy* attribute in an
48983 initialized attributes object referenced by *attr*.

48984 The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new
48985 process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER is set in the *spawn-*
48986 *flags* attribute). The default value of this attribute is unspecified.

48987 **RETURN VALUE**

48988 Upon successful completion, *posix_spawnattr_getschedpolicy()* shall return zero and store the
48989 value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy*
48990 parameter; otherwise, an error number shall be returned to indicate the error.

48991 Upon successful completion, *posix_spawnattr_setschedpolicy()* shall return zero; otherwise, an
48992 error number shall be returned to indicate the error.

48993 **ERRORS**

48994 These functions may fail if:

48995 [EINVAL] The value specified by *attr* is invalid.

48996 The *posix_spawnattr_setschedpolicy()* function may fail if:

48997 [EINVAL] The value of the attribute being set is not valid.

48998 **EXAMPLES**

48999 None.

49000 **APPLICATION USAGE**

49001 These functions are part of the Spawn and Process Scheduling options and need not be provided
49002 on all implementations.

49003 **RATIONALE**

49004 None.

49005 **FUTURE DIRECTIONS**

49006 None.

49007 **SEE ALSO**

49008 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
49009 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
49010 *posix_spawnattr_getsigmask()*

49011 XBD [<sched.h>](#), [<spawn.h>](#)

49012 **CHANGE HISTORY**

49013 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

49014 **NAME**

49015 posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault ‡ get and set the spawn-
49016 sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

49017 **SYNOPSIS**

```
49018 SPN #include <signal.h>
49019 #include <spawn.h>
49020 int posix_spawnattr_getsigdefault(const posix_spawnattr_t
49021 *restrict attr, sigset_t *restrict sigdefault);
49022 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
49023 const sigset_t *restrict sigdefault);
```

49024 **DESCRIPTION**

49025 The *posix_spawnattr_getsigdefault()* function shall obtain the value of the *spawn-sigdefault*
49026 attribute from the attributes object referenced by *attr*.

49027 The *posix_spawnattr_setsigdefault()* function shall set the *spawn-sigdefault* attribute in an
49028 initialized attributes object referenced by *attr*.

49029 The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling
49030 in the new process image (if POSIX_SPAWN_SETSIGDEF is set in the *spawn-flags* attribute) by a
49031 spawn operation. The default value of this attribute shall be an empty signal set.

49032 **RETURN VALUE**

49033 Upon successful completion, *posix_spawnattr_getsigdefault()* shall return zero and store the value
49034 of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter;
49035 otherwise, an error number shall be returned to indicate the error.

49036 Upon successful completion, *posix_spawnattr_setsigdefault()* shall return zero; otherwise, an error
49037 number shall be returned to indicate the error.

49038 **ERRORS**

49039 These functions may fail if:

49040 [EINVAL] The value specified by *attr* is invalid.

49041 The *posix_spawnattr_setsigdefault()* function may fail if:

49042 [EINVAL] The value of the attribute being set is not valid.

49043 **EXAMPLES**

49044 None.

49045 **APPLICATION USAGE**

49046 These functions are part of the Spawn option and need not be provided on all implementations.

49047 **RATIONALE**

49048 None.

49049 **FUTURE DIRECTIONS**

49050 None.

49051 **SEE ALSO**

49052 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*,
49053 *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*

49054 XBD [<signal.h>](#), [<spawn.h>](#)

49055 **CHANGE HISTORY**

49056 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

49057 **NAME**

49058 posix_spawnattr_getsigmask, posix_spawnattr_setsigmask ‡ get and set the spawn-sigmask
 49059 attribute of a spawn attributes object (**ADVANCED REALTIME**)

49060 **SYNOPSIS**

```
49061 SPN #include <signal.h>
49062 #include <spawn.h>
49063 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
49064 sigset_t *restrict sigmask);
49065 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
49066 const sigset_t *restrict sigmask);
```

49067 **DESCRIPTION**

49068 The *posix_spawnattr_getsigmask()* function shall obtain the value of the *spawn-sigmask* attribute
 49069 from the attributes object referenced by *attr*.

49070 The *posix_spawnattr_setsigmask()* function shall set the *spawn-sigmask* attribute in an initialized
 49071 attributes object referenced by *attr*.

49072 The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a
 49073 spawn operation (if *POSIX_SPAWN_SETSIGMASK* is set in the *spawn-flags* attribute). The
 49074 default value of this attribute is unspecified.

49075 **RETURN VALUE**

49076 Upon successful completion, *posix_spawnattr_getsigmask()* shall return zero and store the value
 49077 of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter;
 49078 otherwise, an error number shall be returned to indicate the error.

49079 Upon successful completion, *posix_spawnattr_setsigmask()* shall return zero; otherwise, an error
 49080 number shall be returned to indicate the error.

49081 **ERRORS**

49082 These functions may fail if:

49083 [EINVAL] The value specified by *attr* is invalid.

49084 The *posix_spawnattr_setsigmask()* function may fail if:

49085 [EINVAL] The value of the attribute being set is not valid.

49086 **EXAMPLES**

49087 None.

49088 **APPLICATION USAGE**

49089 These functions are part of the Spawn option and need not be provided on all implementations.

49090 **RATIONALE**

49091 None.

49092 **FUTURE DIRECTIONS**

49093 None.

49094 **SEE ALSO**

49095 *posix_spawn()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*,
 49096 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
 49097 *posix_spawnattr_getschedpolicy()*

49098 XBD [<signal.h>](#), [<spawn.h>](#)

49099 **CHANGE HISTORY**

49100 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

49101 **NAME**

49102 posix_spawnattr_init ‡initialize the spawn attributes object (ADVANCED REALTIME)

49103 **SYNOPSIS**

```
49104 SPN    #include <spawn.h>
49105        int posix_spawnattr_init(posix_spawnattr_t *attr);
```

49106 **DESCRIPTION**

49107 Refer to *posix_spawnattr_destroy()*.

49108 **NAME**

49109 `posix_spawnattr_setflags` ‡ set the spawn-flags attribute of a spawn attributes object
49110 (**ADVANCED REALTIME**)

49111 **SYNOPSIS**

```
49112 SPN    #include <spawn.h>  
49113        int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

49114 **DESCRIPTION**

49115 Refer to [posix_spawnattr_getflags\(\)](#).

49116 **NAME**

49117 posix_spawnattr_setpgroup ‡' set the spawn-pgroup attribute of a spawn attributes object
49118 (ADVANCED REALTIME)

49119 **SYNOPSIS**

```
49120 SPN    #include <spawn.h>  
49121        int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

49122 **DESCRIPTION**

49123 Refer to [posix_spawnattr_getpgroup\(\)](#).

49124 **NAME**

49125 posix_spawnattr_setschedparam ‡ set the spawn-schedparam attribute of a spawn attributes
49126 object (**ADVANCED REALTIME**)

49127 **SYNOPSIS**

```
49128 SPN PS #include <sched.h>  
49129         #include <spawn.h>  
49130         int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
49131         const struct sched_param *restrict schedparam);
```

49132 **DESCRIPTION**

49133 Refer to [posix_spawnattr_getschedparam\(\)](#).

49134 **NAME**

49135 posix_spawnattr_setschedpolicy ¶ set the spawn-schedpolicy attribute of a spawn attributes
49136 object (**ADVANCED REALTIME**)

49137 **SYNOPSIS**

```
49138 SPN PS #include <sched.h>  
49139 #include <spawn.h>  
49140 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
49141 int schedpolicy);
```

49142 **DESCRIPTION**

49143 Refer to [posix_spawnattr_getschedpolicy\(\)](#).

49144 **NAME**

49145 posix_spawnattr_setsigdefault ‡'set the spawn-sigdefault attribute of a spawn attributes object
49146 (ADVANCED REALTIME)

49147 **SYNOPSIS**

```
49148 SPN     #include <signal.h>  
49149         #include <spawn.h>  
  
49150         int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,  
49151                                         const sigset_t *restrict sigdefault);
```

49152 **DESCRIPTION**

49153 Refer to [posix_spawnattr_getsigdefault\(\)](#).

49154 **NAME**

49155 posix_spawnattr_setsigmask ‡'set the spawn-sigmask attribute of a spawn attributes object
49156 (ADVANCED REALTIME)

49157 **SYNOPSIS**

```
49158 SPN    #include <signal.h>  
49159        #include <spawn.h>  
49160        int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
49161        const sigset_t *restrict sigmask);
```

49162 **DESCRIPTION**

49163 Refer to [posix_spawnattr_getsigmask\(\)](#).

49164 **NAME**49165 posix_spawn — spawn a process (**ADVANCED REALTIME**)49166 **SYNOPSIS**

```
49167 SPN #include <spawn.h>
49168 int posix_spawn(pid_t *restrict pid, const char *restrict file,
49169 const posix_spawn_file_actions_t *file_actions,
49170 const posix_spawnattr_t *restrict attrp,
49171 char *const argv[restrict], char *const envp[restrict]);
```

49172 **DESCRIPTION**49173 Refer to *posix_spawn()*.

49174 **NAME**

49175 posix_trace_attr_destroy, posix_trace_attr_init ‡' destroy and initialize the trace stream
 49176 attributes object (**TRACING**)

49177 **SYNOPSIS**

```
49178 OB TRC #include <trace.h>
49179 int posix_trace_attr_destroy(trace_attr_t *attr);
49180 int posix_trace_attr_init(trace_attr_t *attr);
```

49181 **DESCRIPTION**

49182 The *posix_trace_attr_destroy()* function shall destroy an initialized trace attributes object. A
 49183 destroyed *attr* attributes object can be reinitialized using *posix_trace_attr_init()*; the results of
 49184 otherwise referencing the object after it has been destroyed are undefined.

49185 The *posix_trace_attr_init()* function shall initialize a trace attributes object *attr* with the default
 49186 value for all of the individual attributes used by a given implementation. The read-only
 49187 *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object
 49188 shall be set to their appropriate values (see [Section 2.11.1.2](#), on page 542).

49189 Results are undefined if *posix_trace_attr_init()* is called specifying an already initialized *attr*
 49190 attributes object.

49191 Implementations may add extensions to the trace attributes object structure as permitted in XBD
 49192 [Chapter 2](#) (on page 15).

49193 The resulting attributes object (possibly modified by setting individual attributes values), when
 49194 used by *posix_trace_create()*, defines the attributes of the trace stream created. A single attributes
 49195 object can be used in multiple calls to *posix_trace_create()*. After one or more trace streams have
 49196 been created using an attributes object, any function affecting that attributes object, including
 49197 destruction, shall not affect any trace stream previously created. An initialized attributes object
 49198 also serves to receive the attributes of an existing trace stream or trace log when calling the
 49199 *posix_trace_get_attr()* function.

49200 **RETURN VALUE**

49201 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 49202 return the corresponding error number.

49203 **ERRORS**

49204 The *posix_trace_attr_destroy()* function may fail if:

49205 [EINVAL] The value of *attr* is invalid.

49206 The *posix_trace_attr_init()* function shall fail if:

49207 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

49208 **EXAMPLES**

49209 None.

49210 **APPLICATION USAGE**

49211 None.

49212 **RATIONALE**

49213 None.

49214 **FUTURE DIRECTIONS**

49215 The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions may be removed in a future
49216 version.

49217 **SEE ALSO**

49218 *posix_trace_create()*, *posix_trace_get_attr()*, *uname()*

49219 XBD <trace.h>

49220 **CHANGE HISTORY**

49221 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49222 IEEE PASC Interpretation 1003.1 #123 is applied.

49223 **Issue 7**

49224 The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions are marked obsolescent.

49225 **NAME**

49226 posix_trace_attr_getclockres, posix_trace_attr_getcreatetime, posix_trace_attr_getgenversion,
49227 posix_trace_attr_getname, posix_trace_attr_setname — retrieve and set information about a
49228 trace stream (**TRACING**)

49229 **SYNOPSIS**

```
49230 OB TRC #include <time.h>
49231 #include <trace.h>
49232 int posix_trace_attr_getclockres(const trace_attr_t *attr,
49233     struct timespec *resolution);
49234 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
49235     struct timespec *createtime);
49236 #include <trace.h>
49237 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
49238     char *genversion);
49239 int posix_trace_attr_getname(const trace_attr_t *attr,
49240     char *tracename);
49241 int posix_trace_attr_setname(trace_attr_t *attr,
49242     const char *tracename);
```

49243 **DESCRIPTION**

49244 The *posix_trace_attr_getclockres()* function shall copy the clock resolution of the clock used to
49245 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the
49246 *attr* argument into the structure pointed to by the *resolution* argument.

49247 The *posix_trace_attr_getcreatetime()* function shall copy the trace stream creation time from the
49248 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure
49249 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of
49250 creation of the trace stream.

49251 The *posix_trace_attr_getgenversion()* function shall copy the string containing version information
49252 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into
49253 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of
49254 a character array which can store at least {TRACE_NAME_MAX} characters.

49255 The *posix_trace_attr_getname()* function shall copy the string containing the trace name from the
49256 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string
49257 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character
49258 array which can store at least {TRACE_NAME_MAX} characters.

49259 The *posix_trace_attr_setname()* function shall set the name in the *trace-name* attribute of the
49260 attributes object pointed to by the *attr* argument, using the trace name string supplied by the
49261 *tracename* argument. If the supplied string contains more than {TRACE_NAME_MAX}
49262 characters, the name copied into the *trace-name* attribute may be truncated to one less than the
49263 length of {TRACE_NAME_MAX} characters. The default value is a null string.

49264 **RETURN VALUE**

49265 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
49266 return the corresponding error number.

49267 If successful, the *posix_trace_attr_getclockres()* function stores the *clock-resolution* attribute value in
49268 the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

49269 If successful, the *posix_trace_attr_getcreatetime()* function stores the trace stream creation time in

49270 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

49271 If successful, the *posix_trace_attr_getgenversion()* function stores the trace version information in
49272 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

49273 If successful, the *posix_trace_attr_getname()* function stores the trace name in the string pointed
49274 to by *tracename*. Otherwise, the content of this string is unspecified.

49275 ERRORS

49276 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
49277 and *posix_trace_attr_getname()* functions may fail if:

49278 [EINVAL] The value specified by one of the arguments is invalid.

49279 EXAMPLES

49280 None.

49281 APPLICATION USAGE

49282 None.

49283 RATIONALE

49284 None.

49285 FUTURE DIRECTIONS

49286 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
49287 *posix_trace_attr_getname()*, and *posix_trace_attr_setname()* functions may be removed in a future
49288 version.

49289 SEE ALSO

49290 *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*, *uname()*

49291 XBD <[time.h](#)>, <[trace.h](#)>

49292 CHANGE HISTORY

49293 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49294 Issue 7

49295 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
49296 *posix_trace_attr_getname()*, and *posix_trace_attr_setname()* functions are marked obsolescent.

49297 **NAME**

49298 posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy,
 49299 posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited,
 49300 posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy — retrieve and set the
 49301 behavior of a trace stream (**TRACING**)

49302 **SYNOPSIS**

```

49303 OB TRC #include <trace.h>
49304 TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
49305 int *restrict inheritancepolicy);
49306 TRL int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
49307 int *restrict logpolicy);
49308 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict
49309 attr, int *restrict streampolicy);
49310 TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
49311 int inheritancepolicy);
49312 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
49313 int logpolicy);
49314 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
49315 int streampolicy);
    
```

49316 **DESCRIPTION**

49317 TRI The *posix_trace_attr_getinherited()* and *posix_trace_attr_setinherited()* functions, respectively, shall
 49318 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the
 49319 *fork()* and *spawn()* operations. The *inheritance* attribute of the attributes object pointed to by the
 49320 *attr* argument shall be set to one of the following values defined by manifest constants in the
 49321 **<trace.h>** header:

49322 **POSIX_TRACE_CLOSE_FOR_CHILD**

49323 After a *fork()* or *spawn()* operation, the child shall not be traced, and tracing of the parent
 49324 shall continue.

49325 **POSIX_TRACE_INHERITED**

49326 After a *fork()* or *spawn()* operation, if the parent is being traced, its child shall be
 49327 concurrently traced using the same trace stream.

49328 The default value for the *inheritance* attribute is **POSIX_TRACE_CLOSE_FOR_CHILD**.

49329 TRL The *posix_trace_attr_getlogfullpolicy()* and *posix_trace_attr_setlogfullpolicy()* functions,
 49330 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the
 49331 attributes object pointed to by the *attr* argument.

49332 The *log-full-policy* attribute shall be set to one of the following values defined by manifest
 49333 constants in the **<trace.h>** header:

49334 **POSIX_TRACE_LOOP**

49335 The trace log shall loop until the associated trace stream is stopped. This policy means that
 49336 when the trace log gets full, the file system shall reuse the resources allocated to the oldest
 49337 trace events that were recorded. In this way, the trace log will always contain the most
 49338 recent trace events flushed.

49339 **POSIX_TRACE_UNTIL_FULL**

49340 The trace stream shall be flushed to the trace log until the trace log is full. This condition can
 49341 be deduced from the *posix_log_full_status* member status (see the **posix_trace_status_info**
 49342 structure defined in **<trace.h>**). The last recorded trace event shall be the
 49343 **POSIX_TRACE_STOP** trace event.

49344 **POSIX_TRACE_APPEND**
 49345 The associated trace stream shall be flushed to the trace log without log size limitation. If
 49346 the application specifies `POSIX_TRACE_APPEND`, the implementation shall ignore the *log-*
 49347 *max-size* attribute.

49348 The default value for the *log-full-policy* attribute is `POSIX_TRACE_LOOP`.

49349 The `posix_trace_attr_getstreamfullpolicy()` and `posix_trace_attr_setstreamfullpolicy()` functions,
 49350 respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute
 49351 of the attributes object pointed to by the *attr* argument.

49352 The *stream-full-policy* attribute shall be set to one of the following values defined by manifest
 49353 constants in the `<trace.h>` header:

49354 **POSIX_TRACE_LOOP**

49355 The trace stream shall loop until explicitly stopped by the `posix_trace_stop()` function. This
 49356 policy means that when the trace stream is full, the trace system shall reuse the resources
 49357 allocated to the oldest trace events recorded. In this way, the trace stream will always
 49358 contain the most recent trace events recorded.

49359 **POSIX_TRACE_UNTIL_FULL**

49360 The trace stream will run until the trace stream resources are exhausted. Then the trace
 49361 stream will stop. This condition can be deduced from `posix_stream_status` and
 49362 `posix_stream_full_status` (see the **posix_trace_status_info** structure defined in `<trace.h>`).
 49363 When this trace stream is read, a `POSIX_TRACE_STOP` trace event shall be reported after
 49364 reporting the last recorded trace event. The trace system shall reuse the resources allocated
 49365 to any trace events already reported—see the `posix_trace_getnext_event()`,
 49366 `posix_trace_trygetnext_event()`, and `posix_trace_timedgetnext_event()` functions for already
 49367 flushed for an active trace stream with log if the Trace Log option is supported; see the
 49368 `posix_trace_flush()` function. The trace system shall restart the trace stream when it is empty
 49369 and may restart it sooner. A `POSIX_TRACE_START` trace event shall be reported before
 49370 reporting the next recorded trace event.

49371 **POSIX_TRACE_FLUSH**

49372 If the Trace Log option is supported, this policy is identical to the
 49373 `POSIX_TRACE_UNTIL_FULL` trace stream full policy except that the trace stream shall be
 49374 flushed regularly as if `posix_trace_flush()` had been explicitly called. Defining this policy for
 49375 an active trace stream without log shall be invalid.

49376 The default value for the *stream-full-policy* attribute shall be `POSIX_TRACE_LOOP` for an active
 49377 trace stream without log.

49378 **TRL** If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be
 49379 `POSIX_TRACE_FLUSH` for an active trace stream with log.

49380 RETURN VALUE

49381 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 49382 return the corresponding error number.

49383 **TRI** If successful, the `posix_trace_attr_getinherited()` function shall store the *inheritance* attribute value
 49384 in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.

49385 **TRL** If successful, the `posix_trace_attr_getlogfullpolicy()` function shall store the *log-full-policy* attribute
 49386 value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.

49387 If successful, the `posix_trace_attr_getstreamfullpolicy()` function shall store the *stream-full-policy*
 49388 attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is
 49389 undefined.

49390 **ERRORS**

49391 These functions may fail if:

49392 [EINVAL] The value specified by at least one of the arguments is invalid.

49393 **EXAMPLES**

49394 None.

49395 **APPLICATION USAGE**

49396 None.

49397 **RATIONALE**

49398 None.

49399 **FUTURE DIRECTIONS**

49400 The following functions:

49401 *posix_trace_attr_getinherited()*
 49402 *posix_trace_attr_getlogfullpolicy()*
 49403 *posix_trace_attr_getstreamfullpolicy()*
 49404 *posix_trace_attr_setinherited()*
 49405 *posix_trace_attr_setlogfullpolicy()*
 49406 *posix_trace_attr_setstreamfullpolicy()*

49407 may be removed in a future version.

49408 **SEE ALSO**

49409 *fork()*, *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*,
 49410 *posix_trace_getnext_event()*, *posix_trace_start()*

49411 XBD <[trace.h](#)>

49412 **CHANGE HISTORY**

49413 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49414 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/39 is applied, adding the TRL and TRC
 49415 margin codes to the *posix_trace_attr_setlogfullpolicy()* function.

49416 **Issue 7**

49417 SD5-XSH-ERN-116 is applied, adding the missing **restrict** keyword to the
 49418 *posix_trace_attr_getstreamfullpolicy()* function declaration.

49419 These functions are marked obsolescent.

49420 **NAME**

49421 posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize,
 49422 posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxusereventsize,
 49423 posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize,
 49424 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

49425 **SYNOPSIS**

```
49426 OB TRC #include <sys/types.h>
49427 #include <trace.h>
49428 TRL int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
49429 size_t *restrict logsize);
49430 int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
49431 size_t *restrict maxdatasize);
49432 int posix_trace_attr_getmaxsystemeventsize(
49433 const trace_attr_t *restrict attr,
49434 size_t *restrict eventsize);
49435 int posix_trace_attr_getmaxusereventsize(
49436 const trace_attr_t *restrict attr,
49437 size_t data_len, size_t *restrict eventsize);
49438 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
49439 size_t *restrict streamsize);
49440 TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
49441 size_t logsize);
49442 int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
49443 size_t maxdatasize);
49444 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
49445 size_t streamsize);
```

49446 **DESCRIPTION**

49447 TRL The *posix_trace_attr_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*
 49448 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by
 49449 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for
 49450 system and user trace events in the trace log. The default value for the *log-max-size* attribute is
 49451 implementation-defined.

49452 The *posix_trace_attr_setlogsize()* function shall set the maximum allowed size, in bytes, in the *log-*
 49453 *max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value
 49454 supplied by the *logsize* argument.

49455 The trace log size shall be used if the *log-full-policy* attribute is set to *POSIX_TRACE_LOOP* or
 49456 *POSIX_TRACE_UNTIL_FULL*. If the *log-full-policy* attribute is set to *POSIX_TRACE_APPEND*,
 49457 the implementation shall ignore the *log-max-size* attribute.

49458 The *posix_trace_attr_getmaxdatasize()* function shall copy the maximum user trace event data
 49459 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*
 49460 argument into the variable pointed to by the *maxdatasize* argument. The default value for the
 49461 *max-data-size* attribute is implementation-defined.

49462 The *posix_trace_attr_getmaxsystemeventsize()* function shall calculate the maximum memory size,
 49463 in bytes, required to store a single system trace event. This value is calculated for the trace
 49464 stream attributes object pointed to by the *attr* argument and is returned in the variable pointed
 49465 to by the *eventsize* argument.

49466 The values returned as the maximum memory sizes of the user and system trace events shall be
 49467 such that if the sum of the maximum memory sizes of a set of the trace events that may be

49468 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace
 49469 stream, the system provides the necessary resources for recording all those trace events, without
 49470 loss.

49471 The *posix_trace_attr_getmaxusereventsize()* function shall calculate the maximum memory size, in
 49472 bytes, required to store a single user trace event generated by a call to *posix_trace_event()* with a
 49473 *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for
 49474 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable
 49475 pointed to by the *eventsize* argument.

49476 The *posix_trace_attr_getstreamsize()* function shall copy the stream size, in bytes, from the *stream-*
 49477 *min-size* attribute of the attributes object pointed to by the *attr* argument into the variable
 49478 pointed to by the *streamsize* argument.

49479 This stream size is the current total memory size reserved for system and user trace events in the
 49480 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The
 49481 stream size refers to memory used to store trace event records. Other stream data (for example,
 49482 trace attribute values) shall not be included in this size.

49483 The *posix_trace_attr_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in
 49484 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size
 49485 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size
 49486 for the user data argument which may be passed to *posix_trace_event()*. The implementation
 49487 shall be allowed to truncate data passed to *trace_user_event* which is longer than *maxdatasize*.

49488 The *posix_trace_attr_setstreamsize()* function shall set the minimum allowed size, in bytes, in the
 49489 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size
 49490 value supplied by the *streamsize* argument.

49491 **RETURN VALUE**

49492 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 49493 return the corresponding error number.

49494 TRL The *posix_trace_attr_getlogsize()* function stores the maximum trace log allowed size in the object
 49495 pointed to by *logsize*, if successful.

49496 The *posix_trace_attr_getmaxdatasize()* function stores the maximum trace event record memory
 49497 size in the object pointed to by *maxdatasize*, if successful.

49498 The *posix_trace_attr_getmaxsystemeventsize()* function stores the maximum memory size to store a
 49499 single system trace event in the object pointed to by *eventsize*, if successful.

49500 The *posix_trace_attr_getmaxusereventsize()* function stores the maximum memory size to store a
 49501 single user trace event in the object pointed to by *eventsize*, if successful.

49502 The *posix_trace_attr_getstreamsize()* function stores the maximum trace stream allowed size in the
 49503 object pointed to by *streamsize*, if successful.

49504 **ERRORS**

49505 These functions may fail if:

49506 [EINVAL] The value specified by one of the arguments is invalid.

49507 **EXAMPLES**

49508 None.

49509 **APPLICATION USAGE**

49510 None.

49511 **RATIONALE**

49512 None.

49513 **FUTURE DIRECTIONS**

49514 The following functions:

49515 *posix_trace_attr_getlogsize()*49516 *posix_trace_attr_getmaxdatasize()*49517 *posix_trace_attr_getmaxsystemeventsize()*49518 *posix_trace_attr_getmaxusereventsize()*49519 *posix_trace_attr_getstreamsize()*49520 *posix_trace_attr_setlogsize()*49521 *posix_trace_attr_setmaxdatasize()*49522 *posix_trace_attr_setstreamsize()*

49523 may be removed in a future version.

49524 **SEE ALSO**49525 *posix_trace_attr_destroy(), posix_trace_create(), posix_trace_event(), posix_trace_get_attr()*49526 XBD [<sys/types.h>](#), [<trace.h>](#)49527 **CHANGE HISTORY**

49528 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49529 **Issue 7**

49530 These functions are marked obsolescent.

49531 **NAME**

49532 posix_trace_attr_getname — retrieve and set information about a trace stream (**TRACING**)

49533 **SYNOPSIS**

```
49534 OB TRC #include <trace.h>
49535         int posix_trace_attr_getname(const trace_attr_t *attr,
49536         char *tracename);
```

49537 **DESCRIPTION**

49538 Refer to *posix_trace_attr_getclockres()*.

49539 **NAME**

49540 `posix_trace_attr_getstreamfullpolicy` — retrieve and set the behavior of a trace stream
49541 (**TRACING**)

49542 **SYNOPSIS**

```
49543 OB TRC #include <trace.h>  
49544 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict  
49545 attr, int *restrict streampolicy);
```

49546 **DESCRIPTION**

49547 Refer to *posix_trace_attr_getinherited()*.

49548 **NAME**

49549 posix_trace_attr_getstreamsize — retrieve and set trace stream size attributes (**TRACING**)

49550 **SYNOPSIS**

```
49551 OB TRC #include <sys/types.h>
49552         #include <trace.h>
49553         int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
49554         size_t *restrict streamsize);
```

49555 **DESCRIPTION**

49556 Refer to *posix_trace_attr_getlogsize()*.

49557 **NAME**

49558 posix_trace_attr_init — initialize the trace stream attributes object (TRACING)

49559 **SYNOPSIS**

49560 OB TRC #include <trace.h>

49561 int posix_trace_attr_init(trace_attr_t *attr);

49562 **DESCRIPTION**49563 Refer to *posix_trace_attr_destroy()*.

49564 **NAME**

49565 posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy ¶ retrieve and set the behavior
49566 of a trace stream (**TRACING**)

49567 **SYNOPSIS**

```
49568 OB TRC #include <trace.h>  
49569 TRI int posix_trace_attr_setinherited(trace_attr_t *attr,  
49570 int inheritancepolicy);  
49571 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,  
49572 int logpolicy);
```

49573 **DESCRIPTION**

49574 Refer to [posix_trace_attr_getinherited\(\)](#).

49575 **NAME**

49576 posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize — retrieve and set trace stream size
49577 attributes (**TRACING**)

49578 **SYNOPSIS**

```
49579 OB TRC  #include <sys/types.h>  
49580          #include <trace.h>  
  
49581 TRL     int posix_trace_attr_setlogsize(trace_attr_t *attr,  
49582         size_t logsize);  
49583 OB TRC  int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,  
49584         size_t maxdatasize);
```

49585 **DESCRIPTION**

49586 Refer to *posix_trace_attr_getlogsize()*.

49587 **NAME**

49588 posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)

49589 **SYNOPSIS**

```
49590 OB TRC #include <trace.h>
49591         int posix_trace_attr_setname(trace_attr_t *attr,
49592         const char *tracename);
```

49593 **DESCRIPTION**

49594 Refer to *posix_trace_attr_getclockres()*.

49595 **NAME**

49596 `posix_trace_attr_setstreamfullpolicy` ¶ retrieve and set the behavior of a trace stream
49597 (**TRACING**)

49598 **SYNOPSIS**

```
49599 OB TRC #include <trace.h>  
49600 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,  
49601 int streampolicy);
```

49602 **DESCRIPTION**

49603 Refer to *posix_trace_attr_getinherited()*.

49604 **NAME**

49605 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

49606 **SYNOPSIS**

```
49607 OB TRC #include <sys/types.h>
49608         #include <trace.h>
49609         int posix_trace_attr_setstreamsize(trace_attr_t *attr,
49610         size_t streamsize);
```

49611 **DESCRIPTION**

49612 Refer to [posix_trace_attr_getlogsize\(\)](#).

49613 **NAME**

49614 posix_trace_clear — clear trace stream and trace log (TRACING)

49615 **SYNOPSIS**

```
49616 OB TRC #include <sys/types.h>
49617         #include <trace.h>
49618         int posix_trace_clear(trace_id_t trid);
```

49619 **DESCRIPTION**

49620 The *posix_trace_clear()* function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create()* function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

49625 All trace events in the trace stream recorded before the call to *posix_trace_clear()* shall be lost. The *posix_stream_full_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix_trace_clear()* call are recorded; the behavior with respect to trace points that may occur during this call is unspecified.

49629 OB TRL If the Trace Log option is supported and the trace stream has been created with a log, the *posix_trace_clear()* function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create_withlog()* function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix_trace_clear()* shall be the same as the first trace event recorded in the active trace stream after the call to *posix_trace_clear()*. The *posix_log_full_status* status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that all trace events that occurred during the *posix_trace_clear()* call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is `POSIX_TRACE_APPEND`, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

49642 **RETURN VALUE**

49643 Upon successful completion, the *posix_trace_clear()* function shall return a value of zero.
49644 Otherwise, it shall return the corresponding error number.

49645 **ERRORS**

49646 The *posix_trace_clear()* function shall fail if:

49647 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream.

49648 **EXAMPLES**

49649 None.

49650 **APPLICATION USAGE**

49651 None.

49652 **RATIONALE**

49653 None.

49654 **FUTURE DIRECTIONS**

49655 The *posix_trace_clear()* function may be removed in a future version.

49656 **SEE ALSO**

49657 *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*

49658 XBD <[sys/types.h](#)>, <[trace.h](#)>

49659 **CHANGE HISTORY**

49660 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49661 IEEE PASC Interpretation 1003.1 #123 is applied.

49662 **Issue 7**

49663 The *posix_trace_clear()* function is marked obsolescent.

49664 **NAME**49665 posix_trace_close, posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)49666 **SYNOPSIS**

```

49667 OB TRC  #include <trace.h>
49668 TRL     int posix_trace_close(trace_id_t trid);
49669         int posix_trace_open(int file_desc, trace_id_t *trid);
49670         int posix_trace_rewind(trace_id_t trid);

```

49671 **DESCRIPTION**

49672 The *posix_trace_close()* function shall deallocate the trace log identifier indicated by *trid*, and all
49673 of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall
49674 fail.

49675 The *posix_trace_open()* function shall allocate the necessary resources and establish the
49676 connection between a trace log identified by the *file_desc* argument and a trace stream identifier
49677 identified by the object pointed to by the *trid* argument. The *file_desc* argument should be a valid
49678 open file descriptor that corresponds to a trace log. The *file_desc* argument shall be open for
49679 reading. The current trace event timestamp, which specifies the timestamp of the trace event that
49680 will be read by the next call to *posix_trace_getnext_event()*, shall be set to the timestamp of the
49681 oldest trace event recorded in the trace log identified by *trid*.

49682 The *posix_trace_open()* function shall return a trace stream identifier in the variable pointed to by
49683 the *trid* argument, that may only be used by the following functions:

49684	<i>posix_trace_close()</i>	<i>posix_trace_get_attr()</i>
49685	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_get_status()</i>
49686	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_getnext_event()</i>
49687	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_rewind()</i>
49688	<i>posix_trace_eventtypelist_rewind()</i>	

49689 In particular, notice that the operations normally used by a trace controller process, such as
49690 *posix_trace_start()*, *posix_trace_stop()*, or *posix_trace_shutdown()*, cannot be invoked using the
49691 trace stream identifier returned by the *posix_trace_open()* function.

49692 The *posix_trace_rewind()* function shall reset the current trace event timestamp, which specifies
49693 the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event()*,
49694 to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

49695 **RETURN VALUE**

49696 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
49697 return the corresponding error number.

49698 If successful, the *posix_trace_open()* function stores the trace stream identifier value in the object
49699 pointed to by *trid*.

49700 **ERRORS**

49701 The *posix_trace_open()* function shall fail if:

49702	[EINTR]	The operation was interrupted by a signal and thus no trace log was opened.
49703	[EINVAL]	The object pointed to by <i>file_desc</i> does not correspond to a valid trace log.

49704 The *posix_trace_close()* and *posix_trace_rewind()* functions may fail if:

49705 [EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

49706 **EXAMPLES**

49707 None.

49708 **APPLICATION USAGE**

49709 None.

49710 **RATIONALE**

49711 None.

49712 **FUTURE DIRECTIONS**

49713 The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions may be removed in

49714 a future version.

49715 **SEE ALSO**

49716 [*posix_trace_get_attr\(\)*](#), [*posix_trace_get_filter\(\)*](#), [*posix_trace_getnext_event\(\)*](#)

49717 XBD [**<trace.h>**](#)

49718 **CHANGE HISTORY**

49719 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49720 IEEE PASC Interpretation 1003.1 #123 is applied.

49721 **Issue 7**

49722 The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions are marked

49723 obsolescent.

49724 **NAME**

49725 posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown ‡
 49726 trace stream initialization, flush, and shutdown from a process (TRACING)

49727 **SYNOPSIS**

```
49728 OB TRC #include <sys/types.h>
49729 #include <trace.h>
49730 int posix_trace_create(pid_t pid,
49731     const trace_attr_t *restrict attr,
49732     trace_id_t *restrict trid);
49733 TRL int posix_trace_create_withlog(pid_t pid,
49734     const trace_attr_t *restrict attr, int file_desc,
49735     trace_id_t *restrict trid);
49736 int posix_trace_flush(trace_id_t trid);
49737 OB TRC int posix_trace_shutdown(trace_id_t trid);
```

49738 **DESCRIPTION**

49739 The *posix_trace_create()* function shall create an active trace stream. It allocates all the resources
 49740 needed by the trace stream being created for tracing the process specified by *pid* in accordance
 49741 with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and
 49742 shall have been initialized by the function *posix_trace_attr_init()* prior to the *posix_trace_create()*
 49743 call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object
 49744 shall be manipulated through a set of functions described in the *posix_trace_attr* family of
 49745 functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the
 49746 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream
 49747 shall be set to the value of the system clock, if the Timers option is not supported, or to the value
 49748 of the CLOCK_REALTIME clock, if the Timers option is supported.

49749 The *pid* argument represents the target process to be traced. If the process executing this function
 49750 does not have appropriate privileges to trace the process identified by *pid*, an error shall be
 49751 returned. If the *pid* argument is zero, the calling process shall be traced.

49752 The *posix_trace_create()* function shall store the trace stream identifier of the new trace stream in
 49753 the object pointed to by the *trid* argument. This trace stream identifier shall be used in
 49754 subsequent calls to control tracing. The *trid* argument may only be used by the following
 49755 functions:

49756	<i>posix_trace_clear()</i>	<i>posix_trace_getnext_event()</i>
49757	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_shutdown()</i>
49758	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_start()</i>
49759	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_stop()</i>
49760	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_timedgetnext_event()</i>
49761	<i>posix_trace_get_attr()</i>	<i>posix_trace_trid_eventid_open()</i>
49762	<i>posix_trace_get_status()</i>	<i>posix_trace_trygetnext_event()</i>

49763 TEF If the Trace Event Filter option is supported, the following additional functions may use the *trid*
 49764 argument:

```
49765 posix_trace_get_filter() posix_trace_set_filter()
```

49766 In particular, notice that the operations normally used by a trace analyzer process, such as
 49767 *posix_trace_rewind()* or *posix_trace_close()*, cannot be invoked using the trace stream identifier
 49768 returned by the *posix_trace_create()* function.

49769 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is supported, its trace event type filter shall be empty.

49771 The *posix_trace_create()* function may be called multiple times from the same or different processes, with the system-wide limit indicated by the runtime invariant value {TRACE_SYS_MAX}, which has the minimum value {_POSIX_TRACE_SYS_MAX}.

49774 The trace stream identifier returned by the *posix_trace_create()* function in the argument pointed to by *trid* is valid only in the process that made the function call. If it is used from another process, that is a child process, in functions defined in POSIX.1-2017, these functions shall return with the error [EINVAL].

49778 TRL The *posix_trace_create_withlog()* function shall be equivalent to *posix_trace_create()*, except that it associates a trace log with this stream. The *file_desc* argument shall be the file descriptor designating the trace log destination. The function shall fail if this file descriptor refers to a file with a file type that is not compatible with the log policy associated with the trace log. The list of the appropriate file types that are compatible with each log policy is implementation-defined.

49783 The *posix_trace_create_withlog()* function shall return in the parameter pointed to by *trid* the trace stream identifier, which uniquely identifies the newly created trace stream, and shall be used in subsequent calls to control tracing. The *trid* argument may only be used by the following functions:

49787	<i>posix_trace_clear()</i>	<i>posix_trace_get_status()</i>
49788	<i>posix_trace_eventid_equal()</i>	<i>posix_trace_getnext_event()</i>
49789	<i>posix_trace_eventid_get_name()</i>	<i>posix_trace_shutdown()</i>
49790	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>posix_trace_start()</i>
49791	<i>posix_trace_eventtypelist_rewind()</i>	<i>posix_trace_stop()</i>
49792	<i>posix_trace_flush()</i>	<i>posix_trace_timedgetnext_event()</i>
49793	<i>posix_trace_get_attr()</i>	<i>posix_trace_trid_eventid_open()</i>

49794 TEF TRL If the Trace Event Filter option is supported, the following additional functions may use the *trid* argument:

49796 *posix_trace_get_filter()* *posix_trace_set_filter()*

49797 TRL In particular, notice that the operations normally used by a trace analyzer process, such as *posix_trace_rewind()* or *posix_trace_close()*, cannot be invoked using the trace stream identifier returned by the *posix_trace_create_withlog()* function.

49800 The *posix_trace_flush()* function shall initiate a flush operation which copies the contents of the trace stream identified by the argument *trid* into the trace log associated with the trace stream at the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this function shall return an error. The termination of the flush operation can be polled by the *posix_trace_get_status()* function. During the flush operation, it shall be possible to trace new trace events up to the point when the trace stream becomes full. After flushing is completed, the space used by the flushed trace events shall be available for tracing new trace events.

49807 If flushing the trace stream causes the resulting trace log to become full, the trace log full policy shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:

49809 POSIX_TRACE_UNTIL_FULL
 49810 The trace events that have not yet been flushed shall be discarded.

49811 POSIX_TRACE_LOOP
 49812 The trace events that have not yet been flushed shall be written to the beginning of the trace
 49813 log, overwriting previous trace events stored there.

49814 POSIX_TRACE_APPEND
 49815 The trace events that have not yet been flushed shall be appended to the trace log.

49816 The *posix_trace_shutdown()* function shall stop the tracing of trace events in the trace stream
 49817 identified by *trid*, as if *posix_trace_stop()* had been invoked. The *posix_trace_shutdown()* function
 49818 shall free all the resources associated with the trace stream.

49819 The *posix_trace_shutdown()* function shall not return until all the resources associated with the
 49820 trace stream have been freed. When the *posix_trace_shutdown()* function returns, the *trid*
 49821 argument becomes an invalid trace stream identifier. A call to this function shall unconditionally
 49822 deallocate the resources regardless of whether all trace events have been retrieved by the
 49823 analyzer process. Any thread blocked on one of the *trace_getnext_event()* functions (which
 49824 specified this *trid*) before this call is unblocked with the error [EINVAL].

49825 If the process exits, invokes a member of the *exec* family of functions, or is terminated, the trace
 49826 streams that the process had created and that have not yet been shut down, shall be
 49827 automatically shut down as if an explicit call were made to the *posix_trace_shutdown()* function.

49828 TRL For an active trace stream with log, when the *posix_trace_shutdown()* function is called, all trace
 49829 events that have not yet been flushed to the trace log shall be flushed, as in the
 49830 *posix_trace_flush()* function, and the trace log shall be closed.

49831 When a trace log is closed, all the information that may be retrieved later from the trace log
 49832 through the trace interface shall have been written to the trace log. This information includes the
 49833 trace attributes, the list of trace event types (with the mapping between trace event names and
 49834 trace event type identifiers), and the trace status.

49835 In addition, unspecified information shall be written to the trace log to allow detection of a valid
 49836 trace log during the *posix_trace_open()* operation.

49837 The *posix_trace_shutdown()* function shall not return until all trace events have been flushed.

49838 RETURN VALUE

49839 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 49840 return the corresponding error number.

49841 TRL The *posix_trace_create()* and *posix_trace_create_withlog()* functions store the trace stream
 49842 identifier value in the object pointed to by *trid*, if successful.

49843 ERRORS

49844 TRL The *posix_trace_create()* and *posix_trace_create_withlog()* functions shall fail if:

49845 [EAGAIN] No more trace streams can be started now. {TRACE_SYS_MAX} has been
 49846 exceeded.

49847 [EINTR] The operation was interrupted by a signal. No trace stream was created.

49848 [EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

49849 [ENOMEM] The implementation does not currently have sufficient memory to create the
 49850 trace stream with the specified parameters.

49851 [EPERM] The caller does not have appropriate privileges to trace the process specified
 49852 by *pid*.

49853 [ESRCH] The *pid* argument does not refer to an existing process.

49854 TRL The *posix_trace_create_withlog()* function shall fail if:

49855 [EBADF] The *file_desc* argument is not a valid file descriptor open for writing.

49856 [EINVAL] The *file_desc* argument refers to a file with a file type that does not support the log policy associated with the trace log.

49857

49858 [ENOSPC] No space left on device. The device corresponding to the argument *file_desc*

49859 does not contain the space required to create this trace log.

49860 TRL The *posix_trace_flush()* and *posix_trace_shutdown()* functions shall fail if:

49861 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream

49862 with log.

49863 [EFBIG] The trace log file has attempted to exceed an implementation-defined

49864 maximum file size.

49865 [ENOSPC] No space left on device.

49866 **EXAMPLES**
49867 None.

49868 **APPLICATION USAGE**
49869 None.

49870 **RATIONALE**
49871 None.

49872 **FUTURE DIRECTIONS**
49873 The *posix_trace_create()*, *posix_trace_create_withlog()*, *posix_trace_flush()*, and
49874 *posix_trace_shutdown()* functions may be removed in a future version.

49875 **SEE ALSO**
49876 *clock_getres()*, *exec*, *posix_trace_attr_destroy()*, *posix_trace_clear()*, *posix_trace_close()*,
49877 *posix_trace_eventid_equal()*, *posix_trace_eventtypelist_getnext_id()*, *posix_trace_get_attr()*,
49878 *posix_trace_get_filter()*, *posix_trace_getnext_event()*, *posix_trace_start()*, *posix_trace_start()*, *time()*
49879 XBD <sys/types.h>, <trace.h>

49880 **CHANGE HISTORY**
49881 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49882 **Issue 7**
49883 These functions are marked obsolescent.

49884 SD5-XSH-ERN-154 is applied, updating the DESCRIPTION to remove the
49885 *posix_trace_trygetnext_event()* function from the list of functions that use the *trid* argument.

49886 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0441 [358] is applied.

49887 **NAME**

49888 posix_trace_event, posix_trace_eventid_open †'trace functions for instrumenting application
49889 code (**TRACING**)

49890 **SYNOPSIS**

```
49891 OB TRC  #include <sys/types.h>
49892         #include <trace.h>
49893
49893         void posix_trace_event(trace_event_id_t event_id,
49894                               const void *restrict data_ptr, size_t data_len);
49895         int  posix_trace_eventid_open(const char *restrict event_name,
49896                                     trace_event_id_t *restrict event_id);
```

49897 **DESCRIPTION**

49898 The *posix_trace_event()* function shall record the *event_id* and the user data pointed to by *data_ptr*
49899 in the trace stream into which the calling process is being traced and in which *event_id* is not
49900 filtered out. If the total size of the user trace event data represented by *data_len* is not greater
49901 than the declared maximum size for user trace event data, then the *truncation-status* attribute of
49902 the trace event recorded is POSIX_TRACE_NOT_TRUNCATED. Otherwise, the user trace event
49903 data is truncated to this declared maximum size and the *truncation-status* attribute of the trace
49904 event recorded is POSIX_TRACE_TRUNCATED_RECORD.

49905 If there is no trace stream created for the process or if the created trace stream is not running, or
49906 if the trace event specified by *event_id* is filtered out in the trace stream, the *posix_trace_event()*
49907 function shall have no effect.

49908 The *posix_trace_eventid_open()* function shall associate a user trace event name with a trace event
49909 type identifier for the calling process. The trace event name is the string pointed to by the
49910 argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
49911 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user
49912 trace event type identifiers that can be defined for any given process is limited by the maximum
49913 value {TRACE_USER_EVENT_MAX}, which has the minimum value
49914 {POSIX_TRACE_USER_EVENT_MAX}.

49915 If the Trace Inherit option is not supported, the *posix_trace_eventid_open()* function shall associate
49916 the user trace event name pointed to by the *event_name* argument with a trace event type
49917 identifier that is unique for the traced process, and is returned in the variable pointed to by the
49918 *event_id* argument. If the user trace event name has already been mapped for the traced process,
49919 then the previously assigned trace event type identifier shall be returned. If the per-process user
49920 trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-
49921 defined POSIX_TRACE_UNNAMED_USEREVENT (see [Table 2-7](#), on page 546) user trace event
49922 shall be returned.

49923 TRI If the Trace Inherit option is supported, the *posix_trace_eventid_open()* function shall associate the
49924 user trace event name pointed to by the *event_name* argument with a trace event type identifier
49925 that is unique for all the processes being traced in this same trace stream, and is returned in the
49926 variable pointed to by the *event_id* argument. If the user trace event name has already been
49927 mapped for the traced processes, then the previously assigned trace event type identifier shall be
49928 returned. If the per-process user trace event name limit represented by
49929 {TRACE_USER_EVENT_MAX} has been reached, the pre-defined
49930 POSIX_TRACE_UNNAMED_USEREVENT ([Table 2-7](#), on page 546) user trace event shall be
49931 returned.

49932 **Note:** The above procedure, together with the fact that multiple processes can only be traced into the
49933 same trace stream by inheritance, ensure that all the processes that are traced into a trace stream
49934 have the same mapping of trace event names to trace event type identifiers.

49935 If there is no trace stream created, the *posix_trace_eventid_open()* function shall store this
 49936 information for future trace streams created for this process.

49937 **RETURN VALUE**

49938 No return value is defined for the *posix_trace_event()* function.

49939 Upon successful completion, the *posix_trace_eventid_open()* function shall return a value of zero.
 49940 Otherwise, it shall return the corresponding error number. The *posix_trace_eventid_open()*
 49941 function stores the trace event type identifier value in the object pointed to by *event_id*, if
 49942 successful.

49943 **ERRORS**

49944 The *posix_trace_eventid_open()* function shall fail if:

49945 [ENAMETOOLONG]

49946 The size of the name pointed to by the *event_name* argument was longer than
 49947 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

49948 **EXAMPLES**

49949 None.

49950 **APPLICATION USAGE**

49951 None.

49952 **RATIONALE**

49953 None.

49954 **FUTURE DIRECTIONS**

49955 The *posix_trace_event()* and *posix_trace_eventid_open()* functions may be removed in a future
 49956 version.

49957 **SEE ALSO**

49958 [Table 2-7](#) (on page 546), *exec*, *posix_trace_eventid_equal()*, *posix_trace_start()*

49959 XBD [<sys/types.h>](#), [<trace.h>](#)

49960 **CHANGE HISTORY**

49961 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

49962 IEEE PASC Interpretation 1003.1 #123 is applied.

49963 IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of
 49964 the *posix_trace_eventid_open()* function and the *event_id* argument.

49965 **Issue 7**

49966 The *posix_trace_event()* and *posix_trace_eventid_open()* functions are marked obsolescent.

49967 **NAME**

49968 posix_trace_eventid_equal, posix_trace_eventid_get_name, posix_trace_trid_eventid_open ‡
 49969 manipulate the trace event type identifier (**TRACING**)

49970 **SYNOPSIS**

```
49971 OB TRC #include <trace.h>
49972 int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
49973 trace_event_id_t event2);
49974 int posix_trace_eventid_get_name(trace_id_t trid,
49975 trace_event_id_t event, char *event_name);
49976 TEF int posix_trace_trid_eventid_open(trace_id_t trid,
49977 const char *restrict event_name,
49978 trace_event_id_t *restrict event);
```

49979 **DESCRIPTION**

49980 The *posix_trace_eventid_equal()* function shall compare the trace event type identifiers *event1* and
 49981 *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the
 49982 trace event type identifiers *event1* and *event2* are from different trace streams, the return value
 49983 shall be unspecified.

49984 The *posix_trace_eventid_get_name()* function shall return, in the argument pointed to by
 49985 *event_name*, the trace event name associated with the trace event type identifier identified by the
 49986 argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The
 49987 name of the trace event shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 49988 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). Successive calls to
 49989 this function with the same trace event type identifier and the same trace stream identifier shall
 49990 return the same event name.

49991 TEF The *posix_trace_trid_eventid_open()* function shall associate a user trace event name with a trace
 49992 event type identifier for a given trace stream. The trace stream is identified by the *trid* argument,
 49993 and it shall be an active trace stream. The trace event name is the string pointed to by the
 49994 argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 49995 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user
 49996 trace event type identifiers that can be defined for any given process is limited by the maximum
 49997 value {TRACE_USER_EVENT_MAX}, which has the minimum value
 49998 {_POSIX_TRACE_USER_EVENT_MAX}.

49999 If the Trace Inherit option is not supported, the *posix_trace_trid_eventid_open()* function shall
 50000 associate the user trace event name pointed to by the *event_name* argument with a trace event
 50001 type identifier that is unique for the process being traced in the trace stream identified by the *trid*
 50002 argument, and is returned in the variable pointed to by the *event* argument. If the user trace
 50003 event name has already been mapped for the traced process, then the previously assigned trace
 50004 event type identifier shall be returned. If the per-process user trace event name limit represented
 50005 by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined
 50006 POSIX_TRACE_UNNAMED_USEREVENT (see [Table 2-7](#), on page 546) user trace event shall be
 50007 returned.

50008 TEF TRI If the Trace Inherit option is supported, the *posix_trace_trid_eventid_open()* function shall
 50009 associate the user trace event name pointed to by the *event_name* argument with a trace event
 50010 type identifier that is unique for all the processes being traced in the trace stream identified by
 50011 the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user
 50012 trace event name has already been mapped for the traced processes, then the previously
 50013 assigned trace event type identifier shall be returned. If the per-process user trace event name
 50014 limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined

50015 POSIX_TRACE_UNNAMED_USEREVENT (see Table 2-7, on page 546) user trace event shall be
 50016 returned.

50017 **RETURN VALUE**

50018 TEF Upon successful completion, the *posix_trace_eventid_get_name()* and
 50019 *posix_trace_trid_eventid_open()* functions shall return a value of zero. Otherwise, they shall return
 50020 the corresponding error number.

50021 The *posix_trace_eventid_equal()* function shall return a non-zero value if *event1* and *event2* are
 50022 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or
 50023 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*
 50024 is invalid, the behavior shall be unspecified.

50025 The *posix_trace_eventid_get_name()* function stores the trace event name value in the object
 50026 pointed to by *event_name*, if successful.

50027 TEF The *posix_trace_trid_eventid_open()* function stores the trace event type identifier value in the
 50028 object pointed to by *event*, if successful.

50029 **ERRORS**

50030 TEF The *posix_trace_eventid_get_name()* and *posix_trace_trid_eventid_open()* functions shall fail if:

50031 [EINVAL] The *trid* argument was not a valid trace stream identifier.

50032 TEF The *posix_trace_trid_eventid_open()* function shall fail if:

50033 TEF [ENAMETOOLONG]

50034 The size of the name pointed to by the *event_name* argument was longer than
 50035 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

50036 The *posix_trace_eventid_get_name()* function shall fail if:

50037 [EINVAL] The trace event type identifier *event* was not associated with any name.

50038 **EXAMPLES**

50039 None.

50040 **APPLICATION USAGE**

50041 None.

50042 **RATIONALE**

50043 None.

50044 **FUTURE DIRECTIONS**

50045 The *posix_trace_eventid_equal()*, *posix_trace_eventid_get_name()*, and
 50046 *posix_trace_trid_eventid_open()* functions may be removed in a future version.

50047 **SEE ALSO**

50048 Table 2-7 (on page 546), *exec*, *posix_trace_event()*, *posix_trace_getnext_event()*

50049 XBD <trace.h>

50050 **CHANGE HISTORY**

50051 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50052 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

50053 **Issue 7**

50054 These functions are marked obsolescent.

50055 **NAME**50056 posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)50057 **SYNOPSIS**

```
50058 OB TRC #include <sys/types.h>
50059         #include <trace.h>
50060         int posix_trace_eventid_open(const char *restrict event_name,
50061         trace_event_id_t *restrict event_id);
```

50062 **DESCRIPTION**50063 Refer to *posix_trace_event()*.

50064 **NAME**

50065 posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty,
50066 posix_trace_eventset_fill, posix_trace_eventset_ismember — manipulate trace event type sets
50067 (**TRACING**)

50068 **SYNOPSIS**

```
50069 OB TRC #include <trace.h>
50070 TEF int posix_trace_eventset_add(trace_event_id_t event_id,
50071 trace_event_set_t *set);
50072 int posix_trace_eventset_del(trace_event_id_t event_id,
50073 trace_event_set_t *set);
50074 int posix_trace_eventset_empty(trace_event_set_t *set);
50075 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
50076 int posix_trace_eventset_ismember(trace_event_id_t event_id,
50077 const trace_event_set_t *restrict set, int *restrict ismember);
```

50078 **DESCRIPTION**

50079 These primitives manipulate sets of trace event types. They operate on data objects addressable
50080 by the application, not on the current trace event filter of any trace stream.

50081 The *posix_trace_eventset_add()* and *posix_trace_eventset_del()* functions, respectively, shall add or
50082 delete the individual trace event type specified by the value of the argument *event_id* to or from
50083 the trace event type set pointed to by the argument *set*. Adding a trace event type already in the
50084 set or deleting a trace event type not in the set shall not be considered an error.

50085 The *posix_trace_eventset_empty()* function shall initialize the trace event type set pointed to by
50086 the *set* argument such that all trace event types defined, both system and user, shall be excluded
50087 from the set.

50088 The *posix_trace_eventset_fill()* function shall initialize the trace event type set pointed to by the
50089 argument *set*, such that the set of trace event types defined by the argument *what* shall be
50090 included in the set. The value of the argument *what* shall consist of one of the following values,
50091 as defined in the **<trace.h>** header:

50092 **POSIX_TRACE_WOPID_EVENTS**

50093 All the process-independent implementation-defined system trace event types are included
50094 in the set.

50095 **POSIX_TRACE_SYSTEM_EVENTS**

50096 All the implementation-defined system trace event types are included in the set, as are those
50097 defined in POSIX.1-2017.

50098 **POSIX_TRACE_ALL_EVENTS**

50099 All trace event types defined, both system and user, are included in the set.

50100 Applications shall call either *posix_trace_eventset_empty()* or *posix_trace_eventset_fill()* at least
50101 once for each object of type **trace_event_set_t** prior to any other use of that object. If such an
50102 object is not initialized in this way, but is nonetheless supplied as an argument to any of the
50103 *posix_trace_eventset_add()*, *posix_trace_eventset_del()*, or *posix_trace_eventset_ismember()* functions,
50104 the results are undefined.

50105 The *posix_trace_eventset_ismember()* function shall test whether the trace event type specified by
50106 the value of the argument *event_id* is a member of the set pointed to by the argument *set*. The
50107 value returned in the object pointed to by *ismember* argument is zero if the trace event type
50108 identifier is not a member of the set and a value different from zero if it is a member of the set.

50109 RETURN VALUE

50110 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
50111 return the corresponding error number.

50112 ERRORS

50113 These functions may fail if:

50114 [EINVAL] The value of one of the arguments is invalid.

50115 EXAMPLES

50116 None.

50117 APPLICATION USAGE

50118 None.

50119 RATIONALE

50120 None.

50121 FUTURE DIRECTIONS

50122 The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`,
50123 `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions may be removed in a
50124 future version.

50125 SEE ALSO

50126 [*posix_trace_eventid_equal\(\)*](#), [*posix_trace_get_filter\(\)*](#)

50127 XBD <[trace.h](#)>

50128 CHANGE HISTORY

50129 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50130 Issue 7

50131 The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`,
50132 `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions are marked obsolescent.

50133 **NAME**

50134 posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind — iterate over a
50135 mapping of trace event types (**TRACING**)

50136 **SYNOPSIS**

```
50137 OB TRC #include <trace.h>
50138 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
50139     trace_event_id_t *restrict event, int *restrict unavailable);
50140 int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

50141 **DESCRIPTION**

50142 The first time *posix_trace_eventtypelist_getnext_id()* is called, the function shall return in the
50143 variable pointed to by *event* the first trace event type identifier of the list of trace events of the
50144 trace stream identified by the *trid* argument. Successive calls to
50145 *posix_trace_eventtypelist_getnext_id()* return in the variable pointed to by *event* the next trace
50146 event type identifier in that same list. Each time a trace event type identifier is successfully
50147 written into the variable pointed to by the *event* argument, the variable pointed to by the
50148 *unavailable* argument shall be set to zero. When no more trace event type identifiers are available,
50149 and so none is returned, the variable pointed to by the *unavailable* argument shall be set to a
50150 value different from zero.

50151 The *posix_trace_eventtypelist_rewind()* function shall reset the next trace event type identifier to
50152 be read to the first trace event type identifier from the list of trace events used in the trace stream
50153 identified by *trid*.

50154 **RETURN VALUE**

50155 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
50156 return the corresponding error number.

50157 The *posix_trace_eventtypelist_getnext_id()* function stores the trace event type identifier value in
50158 the object pointed to by *event*, if successful.

50159 **ERRORS**

50160 These functions shall fail if:
50161 [EINVAL] The *trid* argument was not a valid trace stream identifier.

50162 **EXAMPLES**

50163 None.

50164 **APPLICATION USAGE**

50165 None.

50166 **RATIONALE**

50167 None.

50168 **FUTURE DIRECTIONS**

50169 The *posix_trace_eventtypelist_getnext_id()* and *posix_trace_eventtypelist_rewind()* functions may be
50170 removed in a future version.

50171 **SEE ALSO**

50172 [posix_trace_event\(\)](#), [posix_trace_eventid_equal\(\)](#), [posix_trace_getnext_event\(\)](#)

50173 XBD [<trace.h>](#)

50174 **CHANGE HISTORY**

50175 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50176 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

50177 **Issue 7**

50178 The *posix_trace_eventtypelist_getnext_id()* and *posix_trace_eventtypelist_rewind()* functions are
50179 marked obsolescent.

50180 **NAME**

50181 posix_trace_flush — trace stream flush from a process (**TRACING**)

50182 **SYNOPSIS**

```
50183 OB TRC #include <sys/types.h>
50184         #include <trace.h>
50185 TRL     int posix_trace_flush(trace_id_t trid);
```

50186 **DESCRIPTION**

50187 Refer to *posix_trace_create()*.

50188 **NAME**

50189 posix_trace_get_attr, posix_trace_get_status ‡' retrieve the trace attributes or trace status
50190 (**TRACING**)

50191 **SYNOPSIS**

```
50192 OB TRC #include <trace.h>
50193
50193     int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
50194     int posix_trace_get_status(trace_id_t trid,
50195                               struct posix_trace_status_info *statusinfo);
```

50196 **DESCRIPTION**

50197 The *posix_trace_get_attr()* function shall copy the attributes of the active trace stream identified
50198 TRL by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid*
50199 may represent a pre-recorded trace log.

50200 The *posix_trace_get_status()* function shall return, in the structure pointed to by the *statusinfo*
50201 argument, the current trace status for the trace stream identified by the *trid* argument. These
50202 status values returned in the structure pointed to by *statusinfo* shall have been appropriately
50203 TRL read to ensure that the returned values are consistent. If the Trace Log option is supported and
50204 the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the
50205 completed trace stream.

50206 Each time the *posix_trace_get_status()* function is used, the overrun status of the trace stream
50207 TRL shall be reset to POSIX_TRACE_NO_OVERRUN immediately after the call completes. If the
50208 Trace Log option is supported, the *posix_trace_get_status()* function shall behave the same as
50209 when the option is not supported except for the following differences:

50210 If the *trid* argument refers to a trace stream with log, each time the *posix_trace_get_status()*
50211 function is used, the log overrun status of the trace stream shall be reset to
50212 POSIX_TRACE_NO_OVERRUN and the *flush_error* status shall be reset to zero
50213 immediately after the call completes.

50214 If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the
50215 status of the completed trace stream and the status values of the trace stream shall not be
50216 reset.

50217 **RETURN VALUE**

50218 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
50219 return the corresponding error number.

50220 The *posix_trace_get_attr()* function stores the trace attributes in the object pointed to by *attr*, if
50221 successful.

50222 The *posix_trace_get_status()* function stores the trace status in the object pointed to by *statusinfo*,
50223 if successful.

50224 **ERRORS**

50225 These functions shall fail if:

50226 [EINVAL] The trace stream argument *trid* does not correspond to a valid active trace
50227 stream or a valid trace log.

50228 EXAMPLES

50229 None.

50230 APPLICATION USAGE

50231 None.

50232 RATIONALE

50233 None.

50234 FUTURE DIRECTIONS

50235 The *posix_trace_get_attr()* and *posix_trace_get_status()* functions may be removed in a future
50236 version.

50237 SEE ALSO

50238 [*posix_trace_attr_destroy\(\)*](#), [*posix_trace_close\(\)*](#), [*posix_trace_create\(\)*](#)

50239 XBD [**<trace.h>**](#)

50240 CHANGE HISTORY

50241 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50242 IEEE PASC Interpretation 1003.1 #123 is applied.

50243 Issue 7

50244 The *posix_trace_get_attr()* and *posix_trace_get_status()* functions are marked obsolescent.

50245 **NAME**

50246 posix_trace_get_filter, posix_trace_set_filter — retrieve and set the filter of an initialized trace
50247 stream (**TRACING**)

50248 **SYNOPSIS**

```
50249 OB TRC  #include <trace.h>
50250 TEF      int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
50251         int posix_trace_set_filter(trace_id_t trid,
50252         const trace_event_set_t *set, int how);
```

50253 **DESCRIPTION**

50254 The *posix_trace_get_filter()* function shall retrieve, into the argument pointed to by *set*, the actual
50255 trace event filter from the trace stream specified by *trid*.

50256 The *posix_trace_set_filter()* function shall change the set of filtered trace event types after a trace
50257 stream identified by the *trid* argument is created. This function may be called prior to starting
50258 the trace stream, or while the trace stream is active. By default, if no call is made to
50259 *posix_trace_set_filter()*, all trace events shall be recorded (that is, none of the trace event types are
50260 filtered out).

50261 If this function is called while the trace is in progress, a special system trace event,
50262 POSIX_TRACE_FILTER, shall be recorded in the trace indicating both the old and the new sets
50263 of filtered trace event types (see [Table 2-4](#) (on page 544) and [Table 2-6](#), on page 545).

50264 If the *posix_trace_set_filter()* function is interrupted by a signal, an error shall be returned and the
50265 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

50266 The value of the argument *how* indicates the manner in which the set is to be changed and shall
50267 have one of the following values, as defined in the **<trace.h>** header:

50268 **POSIX_TRACE_SET_EVENTSET**

50269 The resulting set of trace event types to be filtered shall be the trace event type set pointed
50270 to by the argument *set*.

50271 **POSIX_TRACE_ADD_EVENTSET**

50272 The resulting set of trace event types to be filtered shall be the union of the current set and
50273 the trace event type set pointed to by the argument *set*.

50274 **POSIX_TRACE_SUB_EVENTSET**

50275 The resulting set of trace event types to be filtered shall be all trace event types in the
50276 current set that are not in the set pointed to by the argument *set*; that is, remove each
50277 element of the specified set from the current filter.

50278 **RETURN VALUE**

50279 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
50280 return the corresponding error number.

50281 The *posix_trace_get_filter()* function stores the set of filtered trace event types in *set*, if successful.

50282 **ERRORS**

50283 These functions shall fail if:

50284 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream
50285 or the value of the argument pointed to by *set* is invalid.

50286 [EINTR] The operation was interrupted by a signal.

50287 EXAMPLES

50288 None.

50289 APPLICATION USAGE

50290 None.

50291 RATIONALE

50292 None.

50293 FUTURE DIRECTIONS

50294 The *posix_trace_get_filter()* and *posix_trace_set_filter()* functions may be removed in a future
50295 version.

50296 SEE ALSO

50297 [Table 2-4](#) (on page 544), [Table 2-6](#) (on page 545), [posix_trace_eventset_add\(\)](#)

50298 XBD [<trace.h>](#)

50299 CHANGE HISTORY

50300 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50301 IEEE PASC Interpretation 1003.1 #123 is applied.

50302 Issue 7

50303 The *posix_trace_get_filter()* and *posix_trace_set_filter()* functions are marked obsolescent.

50304 **NAME**

50305 posix_trace_get_status — retrieve the trace status (TRACING)

50306 **SYNOPSIS**

```
50307 OB TRC #include <trace.h>
50308         int posix_trace_get_status(trace_id_t trid,
50309         struct posix_trace_status_info *statusinfo);
```

50310 **DESCRIPTION**50311 Refer to [posix_trace_get_attr\(\)](#).

50312 **NAME**

50313 posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event ‡
 50314 retrieve a trace event (**TRACING**)

50315 **SYNOPSIS**

```
50316 OB TRC #include <sys/types.h>
50317 #include <trace.h>

50318 int posix_trace_getnext_event(trace_id_t trid,
50319     struct posix_trace_event_info *restrict event,
50320     void *restrict data, size_t num_bytes,
50321     size_t *restrict data_len, int *restrict unavailable);
50322 int posix_trace_timedgetnext_event(trace_id_t trid,
50323     struct posix_trace_event_info *restrict event,
50324     void *restrict data, size_t num_bytes,
50325     size_t *restrict data_len, int *restrict unavailable,
50326     const struct timespec *restrict abstime);
50327 int posix_trace_trygetnext_event(trace_id_t trid,
50328     struct posix_trace_event_info *restrict event,
50329     void *restrict data, size_t num_bytes,
50330     size_t *restrict data_len, int *restrict unavailable);
```

50331 **DESCRIPTION**

50332 The *posix_trace_getnext_event()* function shall report a recorded trace event either from an active
 50333 **TRL** trace stream without log or a pre-recorded trace stream identified by the *trid* argument. The
 50334 *posix_trace_trygetnext_event()* function shall report a recorded trace event from an active trace
 50335 stream without log identified by the *trid* argument.

50336 The trace event information associated with the recorded trace event shall be copied by the
 50337 function into the structure pointed to by the argument *event* and the data associated with the
 50338 trace event shall be copied into the buffer pointed to by the *data* argument.

50339 The *posix_trace_getnext_event()* function shall block if the *trid* argument identifies an active trace
 50340 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded
 50341 trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero.
 50342 Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different
 50343 from zero.

50344 The *posix_trace_timedgetnext_event()* function shall attempt to get another trace event from an
 50345 active trace stream without log, as in the *posix_trace_getnext_event()* function. However, if no
 50346 trace event is available from the trace stream, the implied wait shall be terminated when the
 50347 timeout specified by the argument *abstime* expires, and the function shall return the error
 50348 [ETIMEDOUT].

50349 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
 50350 clock upon which timeouts are based (that is, when the value of that clock equals or exceeds
 50351 *abstime*), or if the absolute time specified by *abstime* has already passed at the time of the call.

50352 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 50353 be the resolution of the clock on which it is based. The **timespec** data type is defined in the
 50354 **<time.h>** header.

50355 Under no circumstance shall the function fail with a timeout if a trace event is immediately
 50356 available from the trace stream. The validity of the *abstime* argument need not be checked if a
 50357 trace event is immediately available from the trace stream.

50358 The behavior of this function for a pre-recorded trace stream is unspecified.

50359 TRL The `posix_trace_trygetnext_event()` function shall not block. This function shall return an error if
 50360 the `trid` argument identifies a pre-recorded trace stream. If a recorded trace event was reported,
 50361 the variable pointed to by the `unavailable` argument shall be set to zero. Otherwise, if no trace
 50362 event was reported, the variable pointed to by the `unavailable` argument shall be set to a value
 50363 different from zero.

50364 The argument `num_bytes` shall be the size of the buffer pointed to by the `data` argument. The
 50365 argument `data_len` reports to the application the length in bytes of the data record just
 50366 transferred. If `num_bytes` is greater than or equal to the size of the data associated with the trace
 50367 event pointed to by the `event` argument, all the recorded data shall be transferred. In this case,
 50368 the `truncation-status` member of the trace event structure shall be either
 50369 POSIX_TRACE_NOT_TRUNCATED, if the trace event data was recorded without truncation
 50370 while tracing, or POSIX_TRACE_TRUNCATED_RECORD, if the trace event data was truncated
 50371 when it was recorded. If the `num_bytes` argument is less than the length of recorded trace event
 50372 data, the data transferred shall be truncated to a length of `num_bytes`, the value stored in the
 50373 variable pointed to by `data_len` shall be equal to `num_bytes`, and the `truncation-status` member of
 50374 the `event` structure argument shall be set to POSIX_TRACE_TRUNCATED_READ (see the
 50375 `posix_trace_event_info` structure defined in `<trace.h>`).

50376 The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace
 50377 events shall be reported in the order in which they were generated, up to an implementation-
 50378 defined time resolution that causes the ordering of trace events occurring very close to each
 50379 other to be unknown. Once reported, a trace event cannot be reported again from an active trace
 50380 stream. Once a trace event is reported from an active trace stream without log, the trace stream
 50381 shall make the resources associated with that trace event available to record future generated
 50382 trace events.

50383 RETURN VALUE

50384 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 50385 return the corresponding error number.

50386 If successful, these functions store:

50387 The recorded trace event in the object pointed to by `event`

50388 The trace event information associated with the recorded trace event in the object pointed
 50389 to by `data`

50390 The length of this trace event information in the object pointed to by `data_len`

50391 The value of zero in the object pointed to by `unavailable`

50392 ERRORS

50393 These functions shall fail if:

50394 [EINVAL] The trace stream identifier argument `trid` is invalid.

50395 The `posix_trace_getnext_event()` and `posix_trace_timedgetnext_event()` functions shall fail if:

50396 [EINTR] The operation was interrupted by a signal, and so the call had no effect.

50397 The `posix_trace_trygetnext_event()` function shall fail if:

50398 [EINVAL] The trace stream identifier argument `trid` does not correspond to an active
 50399 trace stream.

50400 The *posix_trace_timedgetnext_event()* function shall fail if:

50401 [EINVAL] There is no trace event immediately available from the trace stream, and the
50402 *timeout* argument is invalid.

50403 [ETIMEDOUT] No trace event was available from the trace stream before the specified
50404 timeout *timeout* expired.

50405 EXAMPLES

50406 None.

50407 APPLICATION USAGE

50408 None.

50409 RATIONALE

50410 None.

50411 FUTURE DIRECTIONS

50412 These functions may be removed in a future version.

50413 SEE ALSO

50414 *posix_trace_close()*, *posix_trace_create()*

50415 XBD <*sys/types.h*>, <*trace.h*>

50416 CHANGE HISTORY

50417 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50418 IEEE PASC Interpretation 1003.1 #123 is applied.

50419 Issue 7

50420 The *posix_trace_getnext_event()*, *posix_trace_timedgetnext_event()*, and
50421 *posix_trace_trygetnext_event()* functions are marked obsolescent.

50422 Functionality relating to the Timers option is moved to the Base.

50423 **NAME**

50424 posix_trace_open, posix_trace_rewind — trace log management (TRACING)

50425 **SYNOPSIS**

```
50426 OB TRC  #include <trace.h>
50427 TRL     int posix_trace_open(int file_desc, trace_id_t *trid);
50428         int posix_trace_rewind(trace_id_t trid);
```

50429 **DESCRIPTION**50430 Refer to [posix_trace_close\(\)](#).

50431 **NAME**

50432 posix_trace_set_filter — set filter of an initialized trace stream (**TRACING**)

50433 **SYNOPSIS**

```
50434 OB TRC #include <trace.h>
50435 TEF     int posix_trace_set_filter(trace_id_t trid,
50436     const trace_event_set_t *set, int how);
```

50437 **DESCRIPTION**

50438 Refer to [posix_trace_get_filter\(\)](#).

50439 **NAME**50440 `posix_trace_shutdown` — trace stream shutdown from a process (**TRACING**)50441 **SYNOPSIS**50442 OB TRC `#include <sys/types.h>`50443 `#include <trace.h>`50444 `int posix_trace_shutdown(trace_id_t trid);`50445 **DESCRIPTION**50446 Refer to [*posix_trace_create\(\)*](#).

50447 **NAME**

50448 posix_trace_start, posix_trace_stop †'trace start and stop(TRACEING)

50449 **SYNOPSIS**

```
50450 OB TRC #include <trace.h>
50451         int posix_trace_start(trace_id_t trid);
50452         int posix_trace_stop (trace_id_t trid);
```

50453 **DESCRIPTION**

50454 The *posix_trace_start()* and *posix_trace_stop()* functions, respectively, shall start and stop the trace stream identified by the argument *trid*.

50456 The effect of calling the *posix_trace_start()* function shall be recorded in the trace stream as the POSIX_TRACE_START system trace event and the status of the trace stream shall become POSIX_TRACE_RUNNING. If the trace stream is in progress when this function is called, the POSIX_TRACE_START system trace event shall not be recorded and the trace stream shall continue to run. If the trace stream is full, the POSIX_TRACE_START system trace event shall not be recorded and the status of the trace stream shall not be changed.

50462 The effect of calling the *posix_trace_stop()* function shall be recorded in the trace stream as the POSIX_TRACE_STOP system trace event and the status of the trace stream shall become POSIX_TRACE_SUSPENDED. If the trace stream is suspended when this function is called, the POSIX_TRACE_STOP system trace event shall not be recorded and the trace stream shall remain suspended. If the trace stream is full, the POSIX_TRACE_STOP system trace event shall not be recorded and the status of the trace stream shall not be changed.

50468 **RETURN VALUE**

50469 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

50471 **ERRORS**

50472 These functions shall fail if:

50473 [EINVAL] The value of the argument *trid* does not correspond to an active trace stream and thus no trace stream was started or stopped.

50475 [EINTR] The operation was interrupted by a signal and thus the trace stream was not necessarily started or stopped.

50477 **EXAMPLES**

50478 None.

50479 **APPLICATION USAGE**

50480 None.

50481 **RATIONALE**

50482 None.

50483 **FUTURE DIRECTIONS**

50484 The *posix_trace_start()* and *posix_trace_stop()* functions may be removed in a future version.

50485 **SEE ALSO**

50486 *posix_trace_create()*

50487 XBD <trace.h>

50488 **CHANGE HISTORY**

50489 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

50490 IEEE PASC Interpretation 1003.1 #123 is applied.

50491 **Issue 7**

50492 The *posix_trace_start()* and *posix_trace_stop()* functions are marked obsolescent.

50493 **NAME**

50494 posix_trace_timedgetnext_event — retrieve a trace event (**TRACING**)

50495 **SYNOPSIS**

```
50496 OB TRC #include <sys/types.h>
50497         #include <trace.h>
50498
50498         int posix_trace_timedgetnext_event(trace_id_t trid,
50499         struct posix_trace_event_info *restrict event,
50500         void *restrict data, size_t num_bytes,
50501         size_t *restrict data_len, int *restrict unavailable,
50502         const struct timespec *restrict abstime);
```

50503 **DESCRIPTION**

50504 Refer to [posix_trace_getnext_event\(\)](#).

50505 **NAME**

50506 posix_trace_trid_eventid_open ‡open a trace event type identifier (TRACING)

50507 **SYNOPSIS**

```
50508 OB TRC  #include <trace.h>
50509 TEF     int posix_trace_trid_eventid_open(trace_id_t trid,
50510         const char *restrict event_name,
50511         trace_event_id_t *restrict event);
```

50512 **DESCRIPTION**50513 Refer to *posix_trace_eventid_equal()*.

50514 **NAME**

50515 posix_trace_trygetnext_event — retrieve a trace event (**TRACING**)

50516 **SYNOPSIS**

```
50517 OB TRC #include <sys/types.h>
50518         #include <trace.h>
50519
50519         int posix_trace_trygetnext_event(trace_id_t trid,
50520         struct posix_trace_event_info *restrict event,
50521         void *restrict data, size_t num_bytes,
50522         size_t *restrict data_len, int *restrict unavailable);
```

50523 **DESCRIPTION**

50524 Refer to [posix_trace_getnext_event\(\)](#).

50525 **NAME**

50526 posix_typed_mem_get_info ‡query typed memory information (ADVANCED REALTIME)

50527 **SYNOPSIS**

```
50528 TYM #include <sys/mman.h>
50529
50529 int posix_typed_mem_get_info(int fildev,
50530 struct posix_typed_mem_info *info);
```

50531 **DESCRIPTION**

50532 The *posix_typed_mem_get_info()* function shall return, in the *posix_tmi_length* field of the
 50533 **posix_typed_mem_info** structure pointed to by *info*, the maximum length which may be
 50534 successfully allocated by the typed memory object designated by *fildev*. This maximum length
 50535 shall take into account the flag POSIX_TYPED_MEM_ALLOCATE or
 50536 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified when the typed memory object
 50537 represented by *fildev* was opened. The maximum length is dynamic; therefore, the value
 50538 returned is valid only while the current mapping of the corresponding typed memory pool
 50539 remains unchanged.

50540 If *fildev* represents a typed memory object opened with neither the
 50541 POSIX_TYPED_MEM_ALLOCATE flag nor the POSIX_TYPED_MEM_ALLOCATE_CONTIG
 50542 flag specified, the returned value of *info->posix_tmi_length* is unspecified.

50543 The *posix_typed_mem_get_info()* function may return additional implementation-defined
 50544 information in other fields of the **posix_typed_mem_info** structure pointed to by *info*.

50545 If the memory object specified by *fildev* is not a typed memory object, then the behavior of this
 50546 function is undefined.

50547 **RETURN VALUE**

50548 Upon successful completion, the *posix_typed_mem_get_info()* function shall return zero;
 50549 otherwise, the corresponding error status value shall be returned.

50550 **ERRORS**

50551 The *posix_typed_mem_get_info()* function shall fail if:

50552 [EBADF] The *fildev* argument is not a valid open file descriptor.

50553 [ENODEV] The *fildev* argument is not connected to a memory object supported by this
 50554 function.

50555 This function shall not return an error code of [EINTR].

50556 **EXAMPLES**

50557 None.

50558 **APPLICATION USAGE**

50559 None.

50560 **RATIONALE**

50561 An application that needs to allocate a block of typed memory with length dependent upon the
 50562 amount of memory currently available must either query the typed memory object to obtain the
 50563 amount available, or repeatedly invoke *mmap()* attempting to guess an appropriate length.
 50564 While the latter method is existing practice with *malloc()*, it is awkward and imprecise. The
 50565 *posix_typed_mem_get_info()* function allows an application to immediately determine available
 50566 memory. This is particularly important for typed memory objects that may in some cases be
 50567 scarce resources. Note that when a typed memory pool is a shared resource, some form of
 50568 mutual-exclusion or synchronization may be required while typed memory is being queried and

50569 allocated to prevent race conditions.

50570 The existing *fstat()* function is not suitable for this purpose. We realize that implementations
50571 may wish to provide other attributes of typed memory objects (for example, alignment
50572 requirements, page size, and so on). The *fstat()* function returns a structure which is not
50573 extensible and, furthermore, contains substantial information that is inappropriate for typed
50574 memory objects.

50575 **FUTURE DIRECTIONS**

50576 None.

50577 **SEE ALSO**

50578 *fstat()*, *mmap()*, *posix_typed_mem_open()*

50579 XBD <[sys/mman.h](#)>

50580 **CHANGE HISTORY**

50581 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

50582 **NAME**50583 posix_typed_mem_open ‡'open a typed memory object **ADVANCED REALTIME**)50584 **SYNOPSIS**

```
50585 TYM #include <sys/mman.h>
50586 int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

50587 **DESCRIPTION**

50588 The *posix_typed_mem_open()* function shall establish a connection between the typed memory
 50589 object specified by the string pointed to by *name* and a file descriptor. It shall create an open file
 50590 description that refers to the typed memory object and a file descriptor that refers to that open
 50591 file description. The file descriptor shall be allocated as described in [Section 2.14](#) (on page 549)
 50592 and can be used by other functions to refer to that typed memory object. It is unspecified
 50593 whether the name appears in the file system and is visible to other functions that take
 50594 pathnames as arguments. The *name* argument conforms to the construction rules for a pathname,
 50595 except that the interpretation of <slash> characters other than the leading <slash> character in
 50596 *name* is implementation-defined, and that the length limits for the *name* argument are
 50597 implementation-defined and need not be the same as the pathname limits {PATH_MAX} and
 50598 {NAME_MAX}. If *name* begins with the <slash> character, then processes calling
 50599 *posix_typed_mem_open()* with the same value of *name* shall refer to the same typed memory
 50600 object. If *name* does not begin with the <slash> character, the effect is implementation-defined.

50601 Each typed memory object supported in a system shall be identified by a name which specifies
 50602 not only its associated typed memory pool, but also the path or port by which it is accessed. That
 50603 is, the same typed memory pool accessed via several different ports shall have several different
 50604 corresponding names. The binding between names and typed memory objects is established in
 50605 an implementation-defined manner. Unlike shared memory objects, there is no way within
 50606 POSIX.1-2017 for a program to create a typed memory object.

50607 The value of *tflag* shall determine how the typed memory object behaves when subsequently
 50608 mapped by calls to *mmap()*. At most, one of the following flags defined in **<sys/mman.h>** may
 50609 be specified:

50610 POSIX_TYPED_MEM_ALLOCATE

50611 Allocate on *mmap()*.

50612 POSIX_TYPED_MEM_ALLOCATE_CONTIG

50613 Allocate contiguously on *mmap()*.

50614 POSIX_TYPED_MEM_MAP_ALLOCATABLE

50615 Map on *mmap()*, without affecting allocatability.

50616 If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap()*
 50617 using the returned file descriptor shall result in allocation and mapping of typed memory from
 50618 the specified typed memory pool. The allocated memory may be a contiguous previously
 50619 unallocated area of the typed memory pool or several non-contiguous previously unallocated
 50620 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag
 50621 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the
 50622 returned file descriptor shall result in allocation and mapping of a single contiguous previously
 50623 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process
 50624 address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or
 50625 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the
 50626 returned file descriptor shall map an application-chosen area from the specified typed memory
 50627 pool such that this mapped area becomes unavailable for allocation until unmapped by all
 50628 processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any

50629 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen
 50630 area from the specified typed memory pool without an effect on the availability of that area for
 50631 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it
 50632 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping.
 50633 Appropriate privileges to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag are
 50634 implementation-defined.

50635 If successful, *posix_typed_mem_open()* shall return a file descriptor for the typed memory object.
 50636 The open file description is new, and therefore the file descriptor shall not share it with any other
 50637 processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag
 50638 associated with the new file descriptor shall be cleared.

50639 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,
 50640 *posix_mem_offset()*, *posix_typed_mem_get_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified
 50641 when passed a file descriptor connected to a typed memory object by this function.

50642 The file status flags of the open file description shall be set according to the value of *oflag*.
 50643 Applications shall specify exactly one of the three access mode values described below and
 50644 defined in the `<fcntl.h>` header, as the value of *oflag*.

- 50645 O_RDONLY Open for read access only.
- 50646 O_WRONLY Open for write access only.
- 50647 O_RDWR Open for read or write access.

50648 **RETURN VALUE**

50649 Upon successful completion, the *posix_typed_mem_open()* function shall return a non-negative
 50650 integer representing the file descriptor. Otherwise, it shall return `-1` and set *errno* to indicate the
 50651 error.

50652 **ERRORS**

50653 The *posix_typed_mem_open()* function shall fail if:

- 50654 [EACCES] The typed memory object exists and the permissions specified by *oflag* are
 50655 denied.
- 50656 [EINTR] The *posix_typed_mem_open()* operation was interrupted by a signal.
- 50657 [EINVAL] The flags specified in *tflag* are invalid (more than one of
 50658 POSIX_TYPED_MEM_ALLOCATE,
 50659 POSIX_TYPED_MEM_ALLOCATE_CONTIG, or
 50660 POSIX_TYPED_MEM_MAP_ALLOCATABLE is specified).
- 50661 [EMFILE] All file descriptors available to the process are currently open.
- 50662 [ENFILE] Too many file descriptors are currently open in the system.
- 50663 [ENOENT] The named typed memory object does not exist.
- 50664 [EPERM] The caller lacks appropriate privileges to specify the
 50665 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag in the *tflag* argument.

50666 The *posix_typed_mem_open()* function may fail if:

- 50667 [ENAMETOOLONG] The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems
 50668 that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI
 50669 XSI systems, or has a pathname component that is longer than
 50670

50671 XSI { _POSIX_NAME_MAX } on systems that do not support the XSI option or
50672 longer than { _XOPEN_NAME_MAX } on XSI systems.

50673 **EXAMPLES**

50674 None.

50675 **APPLICATION USAGE**

50676 None.

50677 **RATIONALE**

50678 None.

50679 **FUTURE DIRECTIONS**

50680 None.

50681 **SEE ALSO**

50682 Section 2.14 (on page 549), *close()*, *dup()*, *exec*, *fcntl()*, *fstat()*, *truncate()*, *mmap()*, *msync()*,
50683 *posix_mem_offset()*, *posix_typed_mem_get_info()*, *umask()*

50684 XBD <fcntl.h>, <sys/mman.h>

50685 **CHANGE HISTORY**

50686 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

50687 **Issue 7**

50688 Austin Group Interpretation 1003.1-2001 #143 is applied.

50689 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0442 [119,428] is applied.

50690 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0256 [835], XSH/TC2-2008/0257 [835],
50691 and XSH/TC2-2008/0258 [835] are applied.

50692 **NAME**50693 `pow, powf, powl` \ddagger power function50694 **SYNOPSIS**50695 `#include <math.h>`50696 `double pow(double x, double y);`50697 `float powf(float x, float y);`50698 `long double powl(long double x, long double y);`50699 **DESCRIPTION**

50700 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 50701 conflict between the requirements described here and the ISO C standard is unintentional. This
 50702 volume of POSIX.1-2017 defers to the ISO C standard.

50703 These functions shall compute the value of x raised to the power y , x^y . If x is negative, the
 50704 application shall ensure that y is an integer value.

50705 An application wishing to check for error situations should set *errno* to zero and call
 50706 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 50707 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 50708 zero, an error has occurred.

50709 **RETURN VALUE**50710 Upon successful completion, these functions shall return the value of x raised to the power y .

50711 MX For finite values of $x < 0$, and finite non-integer values of y , a domain error shall occur and
 50712 either a NaN (if representable), or an implementation-defined value shall be returned.

50713 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and
 50714 *powl()* shall return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL, respectively, with the
 50715 same sign as the correct value of the function.

50716 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 50717 and *pow()*, *powf()*, and *powl()* shall return 0.0, or (if IEC 60559 Floating-Point is not supported)
 50718 an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and
 50719 LDBL_MIN, respectively.

50720 CX For $y < 0$, if x is zero, a pole error may occur and *pow()*, *powf()*, and *powl()* shall return
 50721 \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL, respectively. On systems that support the
 50722 IEC 60559 Floating-Point option, if x is ± 0 , a pole error shall occur and *pow()*, *powf()*, and *powl()*
 50723 shall return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL, respectively if y is an odd integer,
 50724 or HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively if y is not an odd integer.

50725 MX If x or y is a NaN, a NaN shall be returned (unless specified elsewhere in this description).

50726 For any value of y (including NaN), if x is +1, 1.0 shall be returned.50727 For any value of x (including NaN), if y is ± 0 , 1.0 shall be returned.50728 For any odd integer value of $y > 0$, if x is ± 0 , ± 0 shall be returned.50729 For $y > 0$ and not an odd integer, if x is ± 0 , +0 shall be returned.50730 If x is -1 , and y is \pm Inf, 1.0 shall be returned.50731 For $|x| < 1$, if y is $-\text{Inf}$, +Inf shall be returned.50732 For $|x| > 1$, if y is $-\text{Inf}$, +0 shall be returned.50733 For $|x| < 1$, if y is +Inf, +0 shall be returned.

50734 For $|x| > 1$, if y is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

50735 For y an odd integer < 0 , if x is $-\text{Inf}$, $-\text{Inf}$ shall be returned.

50736 For $y < 0$ and not an odd integer, if x is $-\text{Inf}$, $+\text{Inf}$ shall be returned.

50737 For y an odd integer > 0 , if x is $-\text{Inf}$, $-\text{Inf}$ shall be returned.

50738 For $y > 0$ and not an odd integer, if x is $-\text{Inf}$, $+\text{Inf}$ shall be returned.

50739 For $y < 0$, if x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

50740 For $y > 0$, if x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

50741 MXX If the correct value would cause underflow, and is representable, a range error may occur and
50742 the correct value shall be returned.

50743 ERRORS

50744 These functions shall fail if:

50745 Domain Error The value of x is negative and y is a finite non-integer.

50746 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
50747 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
50748 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
50749 shall be raised.

50750 MX Pole Error The value of x is zero and y is negative.

50751 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
50752 then *errno* shall be set to [ERANGE]. If the integer expression
50753 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
50754 floating-point exception shall be raised.

50755 Range Error The result overflows.

50756 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
50757 then *errno* shall be set to [ERANGE]. If the integer expression
50758 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
50759 floating-point exception shall be raised.

50760 These functions may fail if:

50761 Pole Error The value of x is zero and y is negative.

50762 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
50763 then *errno* shall be set to [ERANGE]. If the integer expression
50764 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
50765 floating-point exception shall be raised.

50766 Range Error The result underflows.

50767 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
50768 then *errno* shall be set to [ERANGE]. If the integer expression
50769 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
50770 floating-point exception shall be raised.

50771 **EXAMPLES**

50772 None.

50773 **APPLICATION USAGE**

50774 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
50775 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

50776 **RATIONALE**

50777 None.

50778 **FUTURE DIRECTIONS**

50779 None.

50780 **SEE ALSO**50781 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

50782 XBD Section 4.20 (on page 117), <math.h>

50783 **CHANGE HISTORY**

50784 First released in Issue 1. Derived from Issue 1 of the SVID.

50785 **Issue 5**

50786 The DESCRIPTION is updated to indicate how an application should check for an error. This
50787 text was previously published in the APPLICATION USAGE section.

50788 **Issue 6**

50789 The normative text is updated to avoid use of the term “must” for application requirements.

50790 The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

50791 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
50792 revised to align with the ISO/IEC 9899:1999 standard.

50793 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
50794 marked.

50795 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third
50796 paragraph in the RETURN VALUE section.

50797 **Issue 7**

50798 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

50799 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0443 [68], XSH/TC1-2008/0444 [148],
50800 and XSH/TC1-2008/0445 [68] are applied.

50801 **NAME**

50802 pread — read from a file

50803 **SYNOPSIS**

50804 #include <unistd.h>

50805 ssize_t pread(int *fd*, void **buf*, size_t *nbyte*, off_t *offset*);

50806 **DESCRIPTION**

50807 Refer to *read()*.

50808 **NAME**

50809 printf ¶'print formatted output

50810 **SYNOPSIS**

50811 #include <stdio.h>

50812 int printf(const char *restrict *format*, ...);

50813 **DESCRIPTION**

50814 Refer to *fprintf()*.

50815 **NAME**

50816 pselect, select — synchronous I/O multiplexing

50817 **SYNOPSIS**

```

50818 #include <sys/select.h>
50819 int pselect(int nfds, fd_set *restrict readfds,
50820            fd_set *restrict writefds, fd_set *restrict errorfds,
50821            const struct timespec *restrict timeout,
50822            const sigset_t *restrict sigmask);
50823 int select(int nfds, fd_set *restrict readfds,
50824           fd_set *restrict writefds, fd_set *restrict errorfds,
50825           struct timeval *restrict timeout);
50826 void FD_CLR(int fd, fd_set *fdset);
50827 int FD_ISSET(int fd, fd_set *fdset);
50828 void FD_SET(int fd, fd_set *fdset);
50829 void FD_ZERO(fd_set *fdset);

```

50830 **DESCRIPTION**

50831 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the
50832 *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for
50833 reading, are ready for writing, or have an exceptional condition pending, respectively.

50834 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

50835 For the *select()* function, the timeout period is given in seconds and microseconds in an
50836 argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is
50837 given in seconds and nanoseconds in an argument of type **struct timespec**.

50838 The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when
50839 *sigmask* is a null pointer.

50840 Upon successful completion, the *select()* function may modify the object pointed to by the
50841 *timeout* argument.

50842 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal
50843 OB XSR devices, **STREAMS-based files**, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()*
50844 on file descriptors that refer to other types of file is unspecified.

50845 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall
50846 be checked in each set; that is, the descriptors from zero through *nfds*-1 in the descriptor sets
50847 shall be examined.

50848 If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input
50849 specifies the file descriptors to be checked for being ready to read, and on output indicates
50850 which file descriptors are ready to read.

50851 If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input
50852 specifies the file descriptors to be checked for being ready to write, and on output indicates
50853 which file descriptors are ready to write.

50854 If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input
50855 specifies the file descriptors to be checked for error conditions pending, and on output indicates
50856 which file descriptors have error conditions pending.

50857 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to
50858 by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for
50859 reading, ready for writing, or have an error condition pending, respectively, and shall return the
50860 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the

50861 corresponding bit shall be set upon successful completion if it was set on input and the
50862 associated condition is true for that file descriptor.

50863 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()*
50864 function shall block until at least one of the requested operations becomes ready, until the
50865 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the
50866 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null
50867 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified
50868 time interval expires without any requested operation becoming ready, the function shall return.
50869 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block
50870 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*
50871 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

50872 The use of a timeout does not affect any pending timers set up by *alarm()* or *setitimer()*.

50873 Implementations may place limitations on the maximum timeout interval supported. All
50874 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*
50875 argument specifies a timeout interval greater than the implementation-defined maximum value,
50876 the maximum value shall be used as the actual timeout value. Implementations may also place
50877 limitations on the granularity of timeout intervals. If the requested timeout interval requires a
50878 finer granularity than the implementation supports, the actual timeout interval shall be rounded
50879 up to the next supported value.

50880 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the
50881 caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall
50882 restore the signal mask of the calling thread before returning.

50883 A descriptor shall be considered ready for reading when a call to an input function with
50884 O_NONBLOCK clear would not block, whether or not the function would transfer data
50885 successfully. (The function might return data, an end-of-file indication, or an error other than
50886 one indicating that it is blocked, and in each of these cases the descriptor shall be considered
50887 ready for reading.)

50888 A descriptor shall be considered ready for writing when a call to an output function with
50889 O_NONBLOCK clear would not block, whether or not the function would transfer data
50890 successfully.

50891 If a socket has a pending error, it shall be considered to have an exceptional condition pending.
50892 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for
50893 use with a socket, it is protocol-specific except as noted below. For other file types it is
50894 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or
50895 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate
50896 that the descriptor has no exceptional condition pending.

50897 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with
50898 parameters requesting normal and ancillary data, such that the presence of either type shall
50899 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if
50900 the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with
50901 normal data. If the socket is currently listening, then it shall be marked as readable if an
50902 incoming connection request has been received, and a call to the *accept()* function shall complete
50903 without blocking.

50904 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying
50905 an amount of normal data equal to the current value of the SO_SNDLOWAT option for the
50906 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the
50907 connection attempt has either succeeded or failed leaving a pending error, the socket shall be

50908 marked as writable.

50909 A socket shall be considered to have an exceptional condition pending if a receive operation
50910 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would
50911 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag
50912 would be used to read out-of-band data.) A socket shall also be considered to have an
50913 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other
50914 circumstances under which a socket may be considered to have an exceptional condition
50915 pending are protocol-specific and implementation-defined.

50916 If the `readfds`, `writfds`, and `errorfds` arguments are all null pointers and the `timeout` argument is
50917 not a null pointer, the `pselect()` or `select()` function shall block for the time specified, or until
50918 interrupted by a signal. If the `readfds`, `writfds`, and `errorfds` arguments are all null pointers and
50919 the `timeout` argument is a null pointer, the `pselect()` or `select()` function shall block until
50920 interrupted by a signal.

50921 File descriptors associated with regular files shall always select true for ready to read, ready to
50922 write, and error conditions.

50923 On failure, the objects pointed to by the `readfds`, `writfds`, and `errorfds` arguments shall not be
50924 modified. If the timeout interval expires without the specified condition being true for any of the
50925 specified file descriptors, the objects pointed to by the `readfds`, `writfds`, and `errorfds` arguments
50926 shall have all bits set to 0.

50927 File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`,
50928 `FD_SET()`, and `FD_ZERO()`. It is unspecified whether each of these is a macro or a function. If a
50929 macro definition is suppressed in order to access an actual function, or a program defines an
50930 external identifier with any of these names, the behavior is undefined.

50931 `FD_CLR(fd, fdsetp)` shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not
50932 a member of this set, there shall be no effect on the set, nor will an error be returned.

50933 `FD_ISSET(fd, fdsetp)` shall evaluate to non-zero if the file descriptor *fd* is a member of the set
50934 pointed to by *fdsetp*, and shall evaluate to zero otherwise.

50935 `FD_SET(fd, fdsetp)` shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file
50936 descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be
50937 returned.

50938 `FD_ZERO(fdsetp)` shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is
50939 returned if the set is not empty at the time `FD_ZERO()` is invoked.

50940 The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or
50941 equal to `FD_SETSIZE`, or if *fd* is not a valid file descriptor, or if any of the arguments are
50942 expressions with side-effects.

50943 If a thread gets canceled during a `pselect()` call, the signal mask in effect when executing the
50944 registered cleanup functions is either the original signal mask or the signal mask installed as part
50945 of the `pselect()` call.

50946 RETURN VALUE

50947 Upon successful completion, the `pselect()` and `select()` functions shall return the total number of
50948 bits set in the bit masks. Otherwise, `-1` shall be returned, and `errno` shall be set to indicate the
50949 error.

50950 `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` do not return a value. `FD_ISSET()` shall return a non-
50951 zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and
50952 0 otherwise.

50953 **ERRORS**

50954 Under the following conditions, *pselect()* and *select()* shall fail and set *errno* to:

50955 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a
50956 valid open file descriptor.

50957 [EINTR] The function was interrupted while blocked waiting for any of the selected
50958 descriptors to become ready and before the timeout interval expired.

50959 If SA_RESTART has been set for the interrupting signal, it is implementation-
50960 defined whether the function restarts or returns with [EINTR].

50961 [EINVAL] An invalid timeout interval was specified.

50962 [EINVAL] The *nfds* argument is less than 0 or greater than FD_SETSIZE.

50963 OB XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is
50964 linked (directly or indirectly) downstream from a multiplexer.

50965 **EXAMPLES**

50966 None.

50967 **APPLICATION USAGE**

50968 None.

50969 **RATIONALE**

50970 In earlier versions of the Single UNIX Specification, the *select()* function was defined in the
50971 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was
50972 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the
50973 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.
50974 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`
50975 to include `<sys/select.h>`.

50976 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers
50977 can install an additional cancellation handler which resets the signal mask to the expected value.

```
50978 void cleanup(void *arg)
50979 {
50980     sigset_t *ss = (sigset_t *) arg;
50981     pthread_sigmask(SIG_SETMASK, ss, NULL);
50982 }
50983
50984 int call_pselect(int nfds, fd_set *readfds, fd_set *writefds,
50985                fd_set errorfds, const struct timespec *timeout,
50986                const sigset_t *sigmask)
50987 {
50988     sigset_t oldmask;
50989     int result;
50990     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
50991     pthread_cleanup_push(cleanup, &oldmask);
50992     result = pselect(nfds, readfds, writefds, errorfds, timeout, sigmask);
50993     pthread_cleanup_pop(0);
50994     return result;
50995 }
```

50995 **FUTURE DIRECTIONS**

50996 None.

50997 **SEE ALSO**50998 *accept()*, *alarm()*, *connect()*, *fcntl()*, *getitimer()*, *poll()*, *read()*, *recvmsg()*, *sendmsg()*, *write()*50999 XBD [<sys/select.h>](#), [<sys/time.h>](#)51000 **CHANGE HISTORY**

51001 First released in Issue 4, Version 2.

51002 **Issue 5**

51003 Moved from X/OPEN UNIX extension to BASE.

51004 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when
51005 *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or
51006 equal to FD_SETSIZE.51007 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.51008 **Issue 6**51009 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs*
51010 in the *select()* DESCRIPTION to be *readfds* and *writefds*.

51011 Text referring to sockets is added to the DESCRIPTION.

51012 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
51013 marked as part of the XSI STREAMS Option Group.51014 The following new requirements on POSIX implementations derive from alignment with the
51015 Single UNIX Specification:

51016 These functions are now mandatory.

51017 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail
51018 related to sockets semantics is added to the DESCRIPTION.51019 The *select()* function now requires inclusion of [<sys/select.h>](#).51020 The **restrict** keyword is added to the *select()* prototype for alignment with the
51021 ISO/IEC 9899:1999 standard.51022 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/70 is applied, updating the
51023 DESCRIPTION to reference the signal mask in terms of the calling thread rather than the
51024 process.51025 **Issue 7**51026 SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled
51027 during a call to *pselect()*, and adding example code to the RATIONALE.

51028 Functionality relating to the XSI STREAMS option is marked obsolescent.

51029 Functionality relating to the Threads option is moved to the Base.

51030 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0446 [372] is applied.

51031 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0259 [680] is applied.

51032 **NAME**

51033 psiginfo, psignal ‡write signal information to standard error

51034 **SYNOPSIS**

```
51035 CX #include <signal.h>
51036 void psiginfo(const siginfo_t *pinfo, const char *message);
51037 void psignal(int signum, const char *message);
```

51038 **DESCRIPTION**

51039 The *psiginfo()* and *psignal()* functions shall write a language-dependent message associated with
 51040 a signal number to the standard error stream as follows:

51041 First, if *message* is not a null pointer and is not the empty string, the string pointed to by the
 51042 *message* argument shall be written, followed by a <colon> and a <space>.

51043 Then the signal description string associated with *signum* or with the signal indicated by
 51044 *pinfo* shall be written, followed by a <newline>.

51045 For *psiginfo()*, the application shall ensure that the argument *pinfo* references a valid **siginfo_t**
 51046 structure. For *psignal()*, if *signum* is not a valid signal number, the behavior is implementation-
 51047 defined.

51048 The *psiginfo()* and *psignal()* functions shall not change the orientation of the standard error
 51049 stream.

51050 The *psiginfo()* and *psignal()* functions shall mark for update the last data modification and last
 51051 file status change timestamps of the file associated with the standard error stream at some time
 51052 between their successful completion and *exit()*, *abort()*, or the completion of *fflush()* or *fclose()*
 51053 on *stderr*.

51054 The *psiginfo()* and *psignal()* functions shall not change the setting of *errno* if successful.

51055 On error, the *psiginfo()* and *psignal()* functions shall set the error indicator for the stream to
 51056 which *stderr* points, and shall set *errno* to indicate the error.

51057 Since no value is returned, an application wishing to check for error situations should set *errno*
 51058 to 0, then call *psiginfo()* or *psignal()*, then check *errno*.

51059 **RETURN VALUE**

51060 These functions shall not return a value.

51061 **ERRORS**

51062 Refer to *fputc()*.

51063 **EXAMPLES**

51064 None.

51065 **APPLICATION USAGE**

51066 As an alternative to setting *errno* to zero before the call and checking if it is non-zero afterwards,
 51067 applications can use *ferror()* to detect whether *psiginfo()* or *psignal()* encountered an error.

51068 An application wishing to use this method to check for error situations should call
 51069 *clearerr(stderr)* before calling *psiginfo()* or *psignal()*, then if *ferror(stderr)* returns non-zero, the
 51070 value of *errno* indicates which error occurred.

51071 **RATIONALE**

51072 System V historically has *psignal()* and *psiginfo()* in `<siginfo.h>`. However, the `<siginfo.h>`
51073 header is not specified in the Base Definitions volume of POSIX.1-2017, and the type `siginfo_t` is
51074 defined in `<signal.h>`.

51075 **FUTURE DIRECTIONS**

51076 None.

51077 **SEE ALSO**

51078 *fputc()*, *perror()*, *strsignal()*

51079 XBD `<signal.h>`

51080 **CHANGE HISTORY**

51081 First released in Issue 7.

51082 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0447 [399,428], XSH/TC1-2008/0448
51083 [399], and XSH/TC1-2008/0449 [399,401] are applied.

51084 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0260 [629] is applied.

51085 **NAME**

51086 pthread_atfork — register fork handlers

51087 **SYNOPSIS**

51088 #include <pthread.h>

51089 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
51090 void (*child)(void));51091 **DESCRIPTION**

51092 The *pthread_atfork()* function shall declare fork handlers to be called before and after *fork()*, in
51093 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*
51094 processing commences. The *parent* fork handler shall be called after *fork()* processing completes
51095 in the parent process. The *child* fork handler shall be called after *fork()* processing completes in
51096 the child process. If no handling is desired at one or more of these three points, the
51097 corresponding fork handler address(es) may be set to NULL.

51098 If a *fork()* call in a multi-threaded process leads to a *child* fork handler calling any function that is
51099 not async-signal-safe, the behavior is undefined.

51100 The order of calls to *pthread_atfork()* is significant. The *parent* and *child* fork handlers shall be
51101 called in the order in which they were established by calls to *pthread_atfork()*. The *prepare* fork
51102 handlers shall be called in the opposite order.

51103 **RETURN VALUE**

51104 Upon successful completion, *pthread_atfork()* shall return a value of zero; otherwise, an error
51105 number shall be returned to indicate the error.

51106 **ERRORS**

51107 The *pthread_atfork()* function shall fail if:

51108 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

51109 The *pthread_atfork()* function shall not return an error code of [EINTR].

51110 **EXAMPLES**

51111 None.

51112 **APPLICATION USAGE**

51113 The original usage pattern envisaged for *pthread_atfork()* was for the *prepare* fork handler to lock
51114 mutexes and other locks, and for the *parent* and *child* handlers to unlock them. However, since all
51115 of the relevant unlocking functions, except *sem_post()*, are not async-signal-safe, this usage
51116 results in undefined behavior in the child process unless the only such unlocking function it calls
51117 is *sem_post()*.

51118 **RATIONALE**

51119 There are at least two serious problems with the semantics of *fork()* in a multi-threaded
51120 program. One problem has to do with state (for example, memory) covered by mutexes.
51121 Consider the case where one thread has a mutex locked and the state covered by that mutex is
51122 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state
51123 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply
51124 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about
51125 how to correct or otherwise deal with the inconsistent state in the child.

51126 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the
51127 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe
51128 library routines are promised to be available.

51129 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application

51130 programs may not be aware that a multi-threaded library is in use, and they feel free to call any
51131 number of library routines between the `fork()` and `exec` calls, just as they always have. Indeed,
51132 they may be extant single-threaded programs and cannot, therefore, be expected to obey new
51133 restrictions imposed by the threads library.

51134 On the other hand, the multi-threaded library needs a way to protect its internal state during
51135 `fork()` in case it is re-entered later in the child process. The problem arises especially in multi-
51136 threaded I/O libraries, which are almost sure to be invoked between the `fork()` and `exec` calls to
51137 effect I/O redirection. The solution may require locking mutex variables during `fork()`, or it may
51138 entail simply resetting the state in the child after the `fork()` processing completes.

51139 The `pthread_atfork()` function was intended to provide multi-threaded libraries with a means to
51140 protect themselves from innocent application programs that call `fork()`, and to provide multi-
51141 threaded application programs with a standard mechanism for protecting themselves from
51142 `fork()` calls in a library routine or the application itself.

51143 The expected usage was that the prepare handler would acquire all mutex locks and the other
51144 two fork handlers would release them.

51145 For example, an application could have supplied a prepare routine that acquires the necessary
51146 mutexes the library maintains and supplied child and parent routines that release those
51147 mutexes, thus ensuring that the child would have got a consistent snapshot of the state of the
51148 library (and that no mutexes would have been left stranded). This is good in theory, but in
51149 reality not practical. Each and every mutex and lock in the process must be located and locked.
51150 Every component of a program including third-party components must participate and they
51151 must agree who is responsible for which mutex or lock. This is especially problematic for
51152 mutexes and locks in dynamically allocated memory. All mutexes and locks internal to the
51153 implementation must be locked, too. This possibly delays the thread calling `fork()` for a long
51154 time or even indefinitely since uses of these synchronization objects may not be under control of
51155 the application. A final problem to mention here is the problem of locking streams. At least the
51156 streams under control of the system (like `stdin`, `stdout`, `stderr`) must be protected by locking the
51157 stream with `flockfile()`. But the application itself could have done that, possibly in the same
51158 thread calling `fork()`. In this case, the process will deadlock.

51159 Alternatively, some libraries might have been able to supply just a `child` routine that reinitializes
51160 the mutexes in the library and all associated states to some known value (for example, what it
51161 was when the image was originally executed). This approach is not possible, though, because
51162 implementations are allowed to fail `*_init()` and `*_destroy()` calls for mutexes and locks if the
51163 mutex or lock is still locked. In this case, the `child` routine is not able to reinitialize the mutexes
51164 and locks.

51165 When `fork()` is called, only the calling thread is duplicated in the child process. Synchronization
51166 variables remain in the same state in the child as they were in the parent at the time `fork()` was
51167 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child
51168 process, and any associated states may be inconsistent. The intention was that the parent process
51169 could have avoided this by explicit code that acquires and releases locks critical to the child via
51170 `pthread_atfork()`. In addition, any critical threads would have needed to be recreated and
51171 reinitialized to the proper state in the child (also via `pthread_atfork()`).

51172 A higher-level package may acquire locks on its own data structures before invoking lower-level
51173 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of
51174 initialization for avoiding package deadlock: a package initializes all packages on which it
51175 depends before it calls the `pthread_atfork()` function for itself.

51176 As explained, there is no suitable solution for functionality which requires non-atomic
51177 operations to be protected through mutexes and locks. This is why the POSIX.1 standard since

51178 the 1996 release requires that the child process after *fork()* in a multi-threaded process only calls
51179 async-signal-safe interfaces.

51180 FUTURE DIRECTIONS

51181 The *pthread_atfork()* function may be formally deprecated (for example, by shading it OB) in a
51182 future version of this standard.

51183 SEE ALSO

51184 *atexit()*, *exec*, *fork()*

51185 XBD [<pthread.h>](#), [<sys/types.h>](#)

51186 CHANGE HISTORY

51187 First released in Issue 5. Derived from the POSIX Threads Extension.

51188 IEEE PASC Interpretation 1003.1c #4 is applied.

51189 Issue 6

51190 The *pthread_atfork()* function is marked as part of the Threads option.

51191 The [<pthread.h>](#) header is added to the SYNOPSIS.

51192 Issue 7

51193 The *pthread_atfork()* function is moved from the Threads option to the Base.

51194 SD5-XSH-ERN-145 is applied, updating the RATIONALE.

51195 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0261 [858] is applied.

51196 NAME

51197 pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

51198 SYNOPSIS

```
51199 #include <pthread.h>
51200 int pthread_attr_destroy(pthread_attr_t *attr);
51201 int pthread_attr_init(pthread_attr_t *attr);
```

51202 DESCRIPTION

51203 The *pthread_attr_destroy()* function shall destroy a thread attributes object. An implementation
51204 may cause *pthread_attr_destroy()* to set *attr* to an implementation-defined invalid value. A
51205 destroyed *attr* attributes object can be reinitialized using *pthread_attr_init()*; the results of
51206 otherwise referencing the object after it has been destroyed are undefined.

51207 The *pthread_attr_init()* function shall initialize a thread attributes object *attr* with the default
51208 value for all of the individual attributes used by a given implementation.

51209 The resulting attributes object (possibly modified by setting individual attribute values) when
51210 used by *pthread_create()* defines the attributes of the thread created. A single attributes object can
51211 be used in multiple simultaneous calls to *pthread_create()*. Results are undefined if
51212 *pthread_attr_init()* is called specifying an already initialized *attr* attributes object.

51213 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_destroy()*
51214 does not refer to an initialized thread attributes object.

51215 RETURN VALUE

51216 Upon successful completion, *pthread_attr_destroy()* and *pthread_attr_init()* shall return a value of
51217 0; otherwise, an error number shall be returned to indicate the error.

51218 ERRORS

51219 The *pthread_attr_init()* function shall fail if:

51220 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

51221 These functions shall not return an error code of [EINTR].

51222 EXAMPLES

51223 None.

51224 APPLICATION USAGE

51225 None.

51226 RATIONALE

51227 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to
51228 support probable future standardization in these areas without requiring that the function itself
51229 be changed.

51230 Attributes objects provide clean isolation of the configurable aspects of threads. For example,
51231 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When
51232 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects
51233 can help by allowing the changes to be isolated in a single place, rather than being spread across
51234 every instance of thread creation.

51235 Attributes objects can be used to set up “classes” of threads with similar attributes; for example,
51236 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes
51237 can be defined in a single place and then referenced wherever threads need to be created.
51238 Changes to “class” decisions become straightforward, and detailed analysis of each
51239 *pthread_create()* call is not required.

51240 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had
51241 been specified as structures, adding new attributes would force recompilation of all multi-
51242 threaded programs when the attributes objects are extended; this might not be possible if
51243 different program components were supplied by different vendors.

51244 Additionally, opaque attributes objects present opportunities for improving performance.
51245 Argument validity can be checked once when attributes are set, rather than each time a thread is
51246 created. Implementations often need to cache kernel objects that are expensive to create.
51247 Opaque attributes objects provide an efficient mechanism to detect when cached objects become
51248 invalid due to attribute changes.

51249 Since assignment is not necessarily defined on a given opaque type, implementation-defined
51250 default values cannot be defined in a portable way. The solution to this problem is to allow
51251 attributes objects to be initialized dynamically by attributes object initialization functions, so that
51252 default values can be supplied automatically by the implementation.

51253 The following proposal was provided as a suggested alternative to the supplied attributes:

- 51254 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to
51255 the initialization routines (*pthread_create()*, *pthread_mutex_init()*, *pthread_cond_init()*). The
51256 parameter containing the flags should be an opaque type for extensibility. If no flags are
51257 set in the parameter, then the objects are created with default characteristics. An
51258 implementation may specify implementation-defined flag values and associated
51259 behavior.
- 51260 2. If further specialization of mutexes and condition variables is necessary, implementations
51261 may specify additional procedures that operate on the **pthread_mutex_t** and
51262 **pthread_cond_t** objects (instead of on attributes objects).

51263 The difficulties with this solution are:

- 51264 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using
51265 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,
51266 application programmers need to know the location of each bit. If bits are set or read by
51267 encapsulation (that is, *get* and *set* functions), then the bitmask is merely an
51268 implementation of attributes objects as currently defined and should not be exposed to
51269 the programmer.
- 51270 2. Many attributes are not Boolean or very small integral values. For example, scheduling
51271 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking
51272 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,
51273 the bitmask can only reasonably control whether particular attributes are set or not, and it
51274 cannot serve as the repository of the value itself. The value needs to be specified as a
51275 function parameter (which is non-extensible), or by setting a structure field (which is non-
51276 opaque), or by *get* and *set* functions (making the bitmask a redundant addition to the
51277 attributes objects).

51278 Stack size is defined as an optional attribute because the very notion of a stack is inherently
51279 machine-dependent. Some implementations may not be able to change the size of the stack, for
51280 example, and others may not need to because stack pages may be discontinuous and can be
51281 allocated and released on demand.

51282 The attribute mechanism has been designed in large measure for extensibility. Future extensions
51283 to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-2017 has
51284 to be done with care so as not to affect binary-compatibility.

51285 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,

51286 may have their size fixed at compile time. This means, for example, a *pthread_create()* in an
51287 implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary
51288 application assumes is valid. This suggests that implementations should maintain a size field in
51289 the attributes object, as well as possibly version information, if extensions in different directions
51290 (possibly by different vendors) are to be accommodated.

51291 If an implementation detects that the value specified by the *attr* argument to
51292 *pthread_attr_destroy()* does not refer to an initialized thread attributes object, it is recommended
51293 that the function should fail and report an [EINVAL] error.

51294 If an implementation detects that the value specified by the *attr* argument to *pthread_attr_init()*
51295 refers to an already initialized thread attributes object, it is recommended that the function
51296 should fail and report an [EBUSY] error.

51297 **FUTURE DIRECTIONS**

51298 None.

51299 **SEE ALSO**

51300 [*pthread_attr_getstacksize\(\)*](#), [*pthread_attr_getdetachstate\(\)*](#), [*pthread_create\(\)*](#)

51301 XBD <[*pthread.h*](#)>

51302 **CHANGE HISTORY**

51303 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51304 **Issue 6**

51305 The *pthread_attr_destroy()* and *pthread_attr_init()* functions are marked as part of the Threads
51306 option.

51307 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already
51308 initialized thread attributes object is undefined.

51309 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/71 is applied, updating the ERRORS
51310 section to add the optional [EINVAL] error for the *pthread_attr_destroy()* function, and the
51311 optional [EBUSY] error for the *pthread_attr_init()* function.

51312 **Issue 7**

51313 The *pthread_attr_destroy()* and *pthread_attr_init()* functions are moved from the Threads option
51314 to the Base.

51315 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51316 results in undefined behavior.

51317 The [EBUSY] error for an already initialized thread attributes object is removed; this condition
51318 results in undefined behavior.

51319 **NAME**

51320 pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

51321 **SYNOPSIS**

```
51322 #include <pthread.h>
51323 int pthread_attr_getdetachstate(const pthread_attr_t *attr,
51324     int *detachstate);
51325 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

51326 **DESCRIPTION**

51327 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread
 51328 is created detached, then use of the ID of the newly created thread by the *pthread_detach()* or
 51329 *pthread_join()* function is an error.

51330 The *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* functions, respectively, shall get
 51331 and set the *detachstate* attribute in the *attr* object.

51332 For *pthread_attr_getdetachstate()*, *detachstate* shall be set to either
 51333 PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

51334 For *pthread_attr_setdetachstate()*, the application shall set *detachstate* to either
 51335 PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

51336 A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in
 51337 the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all
 51338 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute
 51339 shall be PTHREAD_CREATE_JOINABLE.

51340 The behavior is undefined if the value specified by the *attr* argument to
 51341 *pthread_attr_getdetachstate()* or *pthread_attr_setdetachstate()* does not refer to an initialized thread
 51342 attributes object.

51343 **RETURN VALUE**

51344 Upon successful completion, *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* shall
 51345 return a value of 0; otherwise, an error number shall be returned to indicate the error.

51346 The *pthread_attr_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*
 51347 if successful.

51348 **ERRORS**

51349 The *pthread_attr_setdetachstate()* function shall fail if:

51350 [EINVAL] The value of *detachstate* was not valid

51351 These functions shall not return an error code of [EINTR].

51352 **EXAMPLES**51353 **Retrieving the detachstate Attribute**

51354 This example shows how to obtain the *detachstate* attribute of a thread attribute object.

```
51355 #include <pthread.h>
51356 pthread_attr_t thread_attr;
51357 int detachstate;
51358 int rc;
51359 /* code initializing thread_attr */
51360 ...
```

```

51361     rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
51362     if (rc!=0) {
51363         /* handle error */
51364         ...
51365     }
51366     else {
51367         /* legal values for detachstate are:
51368          * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
51369          */
51370         ...
51371     }

```

51372 APPLICATION USAGE

51373 None.

51374 RATIONALE

51375 If an implementation detects that the value specified by the *attr* argument to
51376 *pthread_attr_getdetachstate()* or *pthread_attr_setdetachstate()* does not refer to an initialized thread
51377 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

51378 FUTURE DIRECTIONS

51379 None.

51380 SEE ALSO

51381 [*pthread_attr_destroy\(\)*](#), [*pthread_attr_getstacksize\(\)*](#), [*pthread_create\(\)*](#)

51382 XBD [**<pthread.h>**](#)

51383 CHANGE HISTORY

51384 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51385 Issue 6

51386 The *pthread_attr_setdetachstate()* and *pthread_attr_getdetachstate()* functions are marked as part of
51387 the Threads option.

51388 The normative text is updated to avoid use of the term “must” for application requirements.

51389 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/72 is applied, adding the example to the
51390 EXAMPLES section.

51391 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/73 is applied, updating the ERRORS
51392 section to include the optional [EINVAL] error.

51393 Issue 7

51394 The *pthread_attr_setdetachstate()* and *pthread_attr_getdetachstate()* functions are moved from the
51395 Threads option to the Base.

51396 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51397 results in undefined behavior.

51398 **NAME**

51399 pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize
51400 attribute

51401 **SYNOPSIS**

```
51402 #include <pthread.h>
51403 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
51404                               size_t *restrict guardsize);
51405 int pthread_attr_setguardsize(pthread_attr_t *attr,
51406                               size_t guardsize);
```

51407 **DESCRIPTION**

51408 The *pthread_attr_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This
51409 attribute shall be returned in the *guardsize* parameter.

51410 The *pthread_attr_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new
51411 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard
51412 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard
51413 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

51414 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The
51415 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is
51416 created with guard protection, the implementation allocates extra memory at the overflow end
51417 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows
51418 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

51419 A conforming implementation may round up the value contained in *guardsize* to a multiple of
51420 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation
51421 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize()*
51422 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous
51423 *pthread_attr_setguardsize()* function call.

51424 The default value of the *guardsize* attribute is implementation-defined.

51425 If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread
51426 stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the
51427 implementation. It is the responsibility of the application to manage stack overflow along with
51428 stack allocation and management in this case.

51429 The behavior is undefined if the value specified by the *attr* argument to
51430 *pthread_attr_getguardsize()* or *pthread_attr_setguardsize()* does not refer to an initialized thread
51431 attributes object.

51432 **RETURN VALUE**

51433 If successful, the *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions shall return
51434 zero; otherwise, an error number shall be returned to indicate the error.

51435 **ERRORS**

51436 These functions shall fail if:

51437 [EINVAL] The parameter *guardsize* is invalid.

51438 These functions shall not return an error code of [EINTR].

51439 **EXAMPLES**51440 **Retrieving the guardsize Attribute**

51441 This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```
51442 #include <pthread.h>
51443 pthread_attr_t thread_attr;
51444 size_t guardsize;
51445 int rc;
51446 /* code initializing thread_attr */
51447 ...
51448 rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
51449 if (rc != 0) {
51450     /* handle error */
51451     ...
51452 }
51453 else {
51454     if (guardsize > 0) {
51455         /* a guard area of at least guardsize bytes is provided */
51456         ...
51457     }
51458     else {
51459         /* no guard area provided */
51460         ...
51461     }
51462 }
```

51463 **APPLICATION USAGE**

51464 None.

51465 **RATIONALE**

51466 The *guardsize* attribute is provided to the application for two reasons:

- 51467 1. Overflow protection can potentially result in wasted system resources. An application
51468 that creates a large number of threads, and which knows its threads never overflow their
51469 stack, can save system resources by turning off guard areas.
- 51470 2. When threads allocate large data structures on the stack, large guard areas may be needed
51471 to detect stack overflow.

51472 The default size of the guard area is left implementation-defined since on systems supporting
51473 very large page sizes, the overhead might be substantial if at least one guard page is required by
51474 default.

51475 If an implementation detects that the value specified by the *attr* argument to
51476 *pthread_attr_getguardsize()* or *pthread_attr_setguardsize()* does not refer to an initialized thread
51477 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

51478 **FUTURE DIRECTIONS**

51479 None.

51480 **SEE ALSO**51481 XBD [<pthread.h>](#), [<sys/mman.h>](#)51482 **CHANGE HISTORY**

51483 First released in Issue 5.

51484 **Issue 6**51485 In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the
51486 second error condition.51487 The **restrict** keyword is added to the `pthread_attr_getguardsize()` prototype for alignment with the
51488 ISO/IEC 9899:1999 standard.51489 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/74 is applied, updating the ERRORS
51490 section to remove the [EINVAL] error ("The attribute *attr* is invalid."), and replacing it with the
51491 optional [EINVAL] error.51492 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/76 is applied, adding the example to the
51493 EXAMPLES section.51494 **Issue 7**51495 SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.51496 SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the
51497 guard area is implementation-defined.51498 The `pthread_attr_getguardsize()` and `pthread_attr_setguardsize()` functions are moved from the XSI
51499 option to the Base.51500 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51501 results in undefined behavior.

51502 **NAME**

51503 pthread_attr_getinheritsched, pthread_attr_setinheritsched ‡ get and set the inheritsched
 51504 attribute (**REALTIME THREADS**)

51505 **SYNOPSIS**

```
51506 TPS #include <pthread.h>
51507 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
51508     int *restrict inheritsched);
51509 int pthread_attr_setinheritsched(pthread_attr_t *attr,
51510     int inheritsched);
```

51511 **DESCRIPTION**

51512 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions, respectively, shall
 51513 get and set the *inheritsched* attribute in the *attr* argument.

51514 When the attributes objects are used by *pthread_create()*, the *inheritsched* attribute determines
 51515 how the other scheduling attributes of the created thread shall be set.

51516 The supported values of *inheritsched* shall be:

51517 **PTHREAD_INHERIT_SCHED**

51518 Specifies that the thread scheduling attributes shall be inherited from the creating thread,
 51519 and the scheduling attributes in this *attr* argument shall be ignored.

51520 **PTHREAD_EXPLICIT_SCHED**

51521 Specifies that the thread scheduling attributes shall be set to the corresponding values from
 51522 this attributes object.

51523 The symbols **PTHREAD_INHERIT_SCHED** and **PTHREAD_EXPLICIT_SCHED** are defined in
 51524 the **<pthread.h>** header.

51525 The following thread scheduling attributes defined by POSIX.1-2017 are affected by the
 51526 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and
 51527 scheduling contention scope (*contentionscope*).

51528 The behavior is undefined if the value specified by the *attr* argument to
 51529 *pthread_attr_getinheritsched()* or *pthread_attr_setinheritsched()* does not refer to an initialized
 51530 thread attributes object.

51531 **RETURN VALUE**

51532 If successful, the *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions shall
 51533 return zero; otherwise, an error number shall be returned to indicate the error.

51534 **ERRORS**

51535 The *pthread_attr_setinheritsched()* function shall fail if:

51536 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

51537 The *pthread_attr_setinheritsched()* function may fail if:

51538 [EINVAL] The value of *inheritsched* is not valid.

51539 These functions shall not return an error code of [EINTR].

51540 **EXAMPLES**

51541 None.

51542 **APPLICATION USAGE**51543 After these attributes have been set, a thread can be created with the specified attributes using
51544 *pthread_create()*. Using these routines does not affect the current running thread.51545 See [Section 2.9.4](#) (on page 515) for further details on thread scheduling attributes and their
51546 default settings.51547 **RATIONALE**51548 If an implementation detects that the value specified by the *attr* argument to
51549 *pthread_attr_getinheritsched()* or *pthread_attr_setinheritsched()* does not refer to an initialized
51550 thread attributes object, it is recommended that the function should fail and report an [EINVAL]
51551 error.51552 **FUTURE DIRECTIONS**

51553 None.

51554 **SEE ALSO**51555 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getschedpolicy()*,
51556 *pthread_attr_getschedparam()*, *pthread_create()*51557 XBD [<pthread.h>](#), [<sched.h>](#)51558 **CHANGE HISTORY**

51559 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51560 Marked as part of the Realtime Threads Feature Group.

51561 **Issue 6**51562 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are marked as part
51563 of the Threads and Thread Execution Scheduling options.51564 The [ENOSYS] error condition has been removed as stubs need not be provided if an
51565 implementation does not support the Thread Execution Scheduling option.51566 The **restrict** keyword is added to the *pthread_attr_getinheritsched()* prototype for alignment with
51567 the ISO/IEC 9899:1999 standard.51568 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/75 is applied, clarifying the values of
51569 *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS
51570 section for checking when *attr* refers to an uninitialized thread attribute object.51571 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/77 is applied, adding a reference to
51572 [Section 2.9.4](#) (on page 515) in the APPLICATION USAGE section.51573 **Issue 7**51574 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are marked only as
51575 part of the Thread Execution Scheduling option as the Threads option is now part of the Base.51576 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51577 results in undefined behavior.

51578 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0450 [314] is applied.

51579 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0262 [757] is applied.

51580 **NAME**

51581 pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam
51582 attribute

51583 **SYNOPSIS**

```
51584 #include <pthread.h>
51585 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
51586     struct sched_param *restrict param);
51587 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
51588     const struct sched_param *restrict param);
```

51589 **DESCRIPTION**

51590 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions, respectively, shall
51591 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*
51592 structure are defined in the **<sched.h>** header. For the SCHED_FIFO and SCHED_RR policies,
51593 the only required member of *param* is *sched_priority*.

51594 TSP For the SCHED_SPORADIC policy, the required members of the *param* structure are
51595 *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and
51596 *sched_ss_max_repl*. The specified *sched_ss_repl_period* must be greater than or equal to the
51597 specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.
51598 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
51599 function to succeed; if not, the function shall fail. It is unspecified whether the
51600 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
51601 rounded to align with the resolution of the clock being used.

51602 The behavior is undefined if the value specified by the *attr* argument to
51603 *pthread_attr_getschedparam()* or *pthread_attr_setschedparam()* does not refer to an initialized thread
51604 attributes object.

51605 **RETURN VALUE**

51606 If successful, the *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions shall
51607 return zero; otherwise, an error number shall be returned to indicate the error.

51608 **ERRORS**

51609 The *pthread_attr_setschedparam()* function shall fail if:

51610 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

51611 The *pthread_attr_setschedparam()* function may fail if:

51612 [EINVAL] The value of *param* is not valid.

51613 These functions shall not return an error code of [EINTR].

51614 **EXAMPLES**

51615 None.

51616 **APPLICATION USAGE**

51617 After these attributes have been set, a thread can be created with the specified attributes using
51618 *pthread_create()*. Using these routines does not affect the current running thread.

51619 **RATIONALE**

51620 If an implementation detects that the value specified by the *attr* argument to
51621 *pthread_attr_getschedparam()* or *pthread_attr_setschedparam()* does not refer to an initialized thread
51622 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

51623 **FUTURE DIRECTIONS**

51624 None.

51625 **SEE ALSO**51626 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
51627 *pthread_attr_getschedpolicy()*, *pthread_create()*

51628 XBD <pthread.h>, <sched.h>

51629 **CHANGE HISTORY**

51630 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51631 **Issue 6**51632 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are marked as part
51633 of the Threads option.

51634 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

51635 The **restrict** keyword is added to the *pthread_attr_getschedparam()* and
51636 *pthread_attr_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.51637 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/78 is applied, updating the ERRORS
51638 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
51639 object.51640 **Issue 7**51641 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are moved from the
51642 Threads option to the Base.51643 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
51644 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.51645 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51646 results in undefined behavior.

51647 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0451 [314] is applied.

51648 **NAME**

51649 pthread_attr_getschedpolicy, pthread_attr_setschedpolicy ‡ get and set the schedpolicy
 51650 attribute (**REALTIME THREADS**)

51651 **SYNOPSIS**

```
51652 TPS #include <pthread.h>
51653 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
51654 int *restrict policy);
51655 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

51656 **DESCRIPTION**

51657 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions, respectively, shall
 51658 get and set the *schedpolicy* attribute in the *attr* argument.

51659 The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER,
 51660 which are defined in the **<sched.h>** header. When threads executing with the scheduling policy
 51661 TSP SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire
 51662 the mutex in priority order when the mutex is unlocked.

51663 The behavior is undefined if the value specified by the *attr* argument to
 51664 *pthread_attr_getschedpolicy()* or *pthread_attr_setschedpolicy()* does not refer to an initialized thread
 51665 attributes object.

51666 **RETURN VALUE**

51667 If successful, the *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions shall
 51668 return zero; otherwise, an error number shall be returned to indicate the error.

51669 **ERRORS**

51670 The *pthread_attr_setschedpolicy()* function shall fail if:

51671 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

51672 The *pthread_attr_setschedpolicy()* function may fail if:

51673 [EINVAL] The value of *policy* is not valid.

51674 These functions shall not return an error code of [EINTR].

51675 **EXAMPLES**

51676 None.

51677 **APPLICATION USAGE**

51678 After these attributes have been set, a thread can be created with the specified attributes using
 51679 *pthread_create()*. Using these routines does not affect the current running thread.

51680 See [Section 2.9.4](#) (on page 515) for further details on thread scheduling attributes and their
 51681 default settings.

51682 **RATIONALE**

51683 If an implementation detects that the value specified by the *attr* argument to
 51684 *pthread_attr_getschedpolicy()* or *pthread_attr_setschedpolicy()* does not refer to an initialized thread
 51685 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

51686 **FUTURE DIRECTIONS**

51687 None.

51688 **SEE ALSO**

51689 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
51690 *pthread_attr_getschedparam()*, *pthread_create()*

51691 XBD <pthread.h>, <sched.h>

51692 **CHANGE HISTORY**

51693 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51694 Marked as part of the Realtime Threads Feature Group.

51695 **Issue 6**

51696 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are marked as part of
51697 the Threads and Thread Execution Scheduling options.

51698 The [ENOSYS] error condition has been removed as stubs need not be provided if an
51699 implementation does not support the Thread Execution Scheduling option.

51700 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

51701 The **restrict** keyword is added to the *pthread_attr_getschedpolicy()* prototype for alignment with
51702 the ISO/IEC 9899:1999 standard.

51703 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/79 is applied, adding a reference to
51704 [Section 2.9.4](#) (on page 515) in the APPLICATION USAGE section.

51705 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/80 is applied, updating the ERRORS
51706 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
51707 object.

51708 **Issue 7**

51709 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are marked only as
51710 part of the Thread Execution Scheduling option as the Threads option is now part of the Base.

51711 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51712 results in undefined behavior.

51713 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0452 [314] is applied.

51714 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0263 [757] is applied.

51715 **NAME**

51716 pthread_attr_getscope, pthread_attr_setscope ‡ get and set the contentionscope attribute
 51717 (**REALTIME THREADS**)

51718 **SYNOPSIS**

```
51719 TPS #include <pthread.h>
51720 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
51721 int *restrict contentionscope);
51722 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

51723 **DESCRIPTION**

51724 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions, respectively, shall get and set
 51725 the *contentionscope* attribute in the *attr* object.

51726 The *contentionscope* attribute may have the values `PTHREAD_SCOPE_SYSTEM`, signifying
 51727 system scheduling contention scope, or `PTHREAD_SCOPE_PROCESS`, signifying process
 51728 scheduling contention scope. The symbols `PTHREAD_SCOPE_SYSTEM` and
 51729 `PTHREAD_SCOPE_PROCESS` are defined in the `<pthread.h>` header.

51730 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getscope()* or
 51731 *pthread_attr_setscope()* does not refer to an initialized thread attributes object.

51732 **RETURN VALUE**

51733 If successful, the *pthread_attr_getscope()* and *pthread_attr_setscope()* functions shall return zero;
 51734 otherwise, an error number shall be returned to indicate the error.

51735 **ERRORS**

51736 The *pthread_attr_setscope()* function shall fail if:

51737 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

51738 The *pthread_attr_setscope()* function may fail if:

51739 [EINVAL] The value of *contentionscope* is not valid.

51740 These functions shall not return an error code of [EINTR].

51741 **EXAMPLES**

51742 None.

51743 **APPLICATION USAGE**

51744 After these attributes have been set, a thread can be created with the specified attributes using
 51745 *pthread_create()*. Using these routines does not affect the current running thread.

51746 See [Section 2.9.4](#) (on page 515) for further details on thread scheduling attributes and their
 51747 default settings.

51748 **RATIONALE**

51749 If an implementation detects that the value specified by the *attr* argument to
 51750 *pthread_attr_getscope()* or *pthread_attr_setscope()* does not refer to an initialized thread attributes
 51751 object, it is recommended that the function should fail and report an [EINVAL] error.

51752 **FUTURE DIRECTIONS**

51753 None.

51754 **SEE ALSO**

51755 *pthread_attr_destroy()*, *pthread_attr_getinheritsched()*, *pthread_attr_getschedpolicy()*,
51756 *pthread_attr_getschedparam()*, *pthread_create()*

51757 XBD <pthread.h>, <sched.h>

51758 **CHANGE HISTORY**

51759 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51760 Marked as part of the Realtime Threads Feature Group.

51761 **Issue 6**

51762 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are marked as part of the
51763 Threads and Thread Execution Scheduling options.

51764 The [ENOSYS] error condition has been removed as stubs need not be provided if an
51765 implementation does not support the Thread Execution Scheduling option.

51766 The **restrict** keyword is added to the *pthread_attr_getscope()* prototype for alignment with the
51767 ISO/IEC 9899:1999 standard.

51768 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/81 is applied, adding a reference to
51769 [Section 2.9.4](#) (on page 515) in the APPLICATION USAGE section.

51770 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/82 is applied, updating the ERRORS
51771 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
51772 object.

51773 **Issue 7**

51774 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are marked only as part of the
51775 Thread Execution Scheduling option as the Threads option is now part of the Base.

51776 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51777 results in undefined behavior.

51778 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0453 [314] is applied.

51779 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0264 [757] is applied.

51780 **NAME**

51781 pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

51782 **SYNOPSIS**

```
51783 TSA TSS #include <pthread.h>
51784 int pthread_attr_getstack(const pthread_attr_t *restrict attr,
51785 void **restrict stackaddr, size_t *restrict stacksize);
51786 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
51787 size_t stacksize);
```

51788 **DESCRIPTION**

51789 The *pthread_attr_getstack()* and *pthread_attr_setstack()* functions, respectively, shall get and set the
 51790 thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

51791 The stack attributes specify the area of storage to be used for the created thread's stack. The base
 51792 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be
 51793 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The
 51794 *pthread_attr_setstack()* function may fail with [EINVAL] if *stackaddr* does not meet
 51795 implementation-defined alignment requirements. All pages within the stack described by
 51796 *stackaddr* and *stacksize* shall be both readable and writable by the thread.

51797 If the *pthread_attr_getstack()* function is called before the *stackaddr* attribute has been set, the
 51798 behavior is unspecified.

51799 The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getstack()* or
 51800 *pthread_attr_setstack()* does not refer to an initialized thread attributes object.

51801 **RETURN VALUE**

51802 Upon successful completion, these functions shall return a value of 0; otherwise, an error
 51803 number shall be returned to indicate the error.

51804 The *pthread_attr_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*
 51805 if successful.

51806 **ERRORS**

51807 The *pthread_attr_setstack()* function shall fail if:

51808 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an
 51809 implementation-defined limit.

51810 The *pthread_attr_setstack()* function may fail if:

51811 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or
 51812 ((**char** *)*stackaddr* + *stacksize*) lacks proper alignment.

51813 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable
 51814 and writable by the thread.

51815 These functions shall not return an error code of [EINTR].

51816 **EXAMPLES**

51817 None.

51818 **APPLICATION USAGE**51819 These functions are appropriate for use by applications in an environment where the stack for a
51820 thread must be placed in some particular region of memory.51821 While it might seem that an application could detect stack overflow by providing a protected
51822 page outside the specified stack region, this cannot be done portably. Implementations are free
51823 to place the thread's initial stack pointer anywhere within the specified region to accommodate
51824 the machine's stack pointer behavior and allocation requirements. Furthermore, on some
51825 architectures, such as the IA-64, "overflow" might mean that two separate stack pointers
51826 allocated within the region will overlap somewhere in the middle of the region.51827 After a successful call to *pthread_attr_setstack()*, the storage area specified by the *stackaddr*
51828 parameter is under the control of the implementation, as described in [Section 2.9.8](#) (on page 522).51829 The specification of the *stackaddr* attribute presents several ambiguities that make portable use of
51830 these functions impossible. For example, the standard allows implementations to impose
51831 arbitrary alignment requirements on *stackaddr*. Applications cannot assume that a buffer
51832 obtained from *malloc()* is suitably aligned. Note that although the *stacksize* value passed to
51833 *pthread_attr_setstack()* must satisfy alignment requirements, the same is not true for
51834 *pthread_attr_setstacksize()* where the implementation must increase the specified size if necessary
51835 to achieve the proper alignment.51836 **RATIONALE**51837 If an implementation detects that the value specified by the *attr* argument to
51838 *pthread_attr_getstack()* or *pthread_attr_setstack()* does not refer to an initialized thread attributes
51839 object, it is recommended that the function should fail and report an [EINVAL] error.51840 **FUTURE DIRECTIONS**

51841 None.

51842 **SEE ALSO**51843 *pthread_attr_destroy()*, *pthread_attr_getdetachstate()*, *pthread_attr_getstacksize()*, *pthread_create()*51844 XBD [<limits.h>](#), [<pthread.h>](#)51845 **CHANGE HISTORY**51846 First released in Issue 6. Developed as part of the XSI option and brought into the BASE by IEEE
51847 PASC Interpretation 1003.1 #101.51848 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/83 is applied, updating the
51849 APPLICATION USAGE section to refer to [Section 2.9.8](#) (on page 522).51850 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC/D6/84 is applied, updating the ERRORS
51851 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
51852 object.51853 **Issue 7**51854 SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.51855 Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is
51856 called before the *stackaddr* attribute is set.

51857 SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

51858 The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION
51859 USAGE sections.

51860
51861

The [EINVAL] error for an uninitialized thread attributes object is removed; this condition results in undefined behavior.

51862 **NAME**

51863 pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

51864 **SYNOPSIS**

```
51865 TSS #include <pthread.h>
51866 int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
51867 size_t *restrict stacksize);
51868 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

51869 **DESCRIPTION**

51870 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions, respectively, shall get and
 51871 set the thread creation *stacksize* attribute in the *attr* object.

51872 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created
 51873 threads stack.

51874 The behavior is undefined if the value specified by the *attr* argument to
 51875 *pthread_attr_getstacksize()* or *pthread_attr_setstacksize()* does not refer to an initialized thread
 51876 attributes object.

51877 **RETURN VALUE**

51878 Upon successful completion, *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* shall
 51879 return a value of 0; otherwise, an error number shall be returned to indicate the error.

51880 The *pthread_attr_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if
 51881 successful.

51882 **ERRORS**

51883 The *pthread_attr_setstacksize()* function shall fail if:

51884 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a
 51885 system-imposed limit.

51886 These functions shall not return an error code of [EINTR].

51887 **EXAMPLES**

51888 None.

51889 **APPLICATION USAGE**

51890 None.

51891 **RATIONALE**

51892 If an implementation detects that the value specified by the *attr* argument to
 51893 *pthread_attr_getstacksize()* or *pthread_attr_setstacksize()* does not refer to an initialized thread
 51894 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

51895 **FUTURE DIRECTIONS**

51896 None.

51897 **SEE ALSO**

51898 *pthread_attr_destroy()*, *pthread_attr_getdetachstate()*, *pthread_create()*

51899 XBD <limits.h>, <pthread.h>

51900 **CHANGE HISTORY**

51901 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51902 **Issue 6**

51903 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are marked as part of the
51904 Threads and Thread Stack Size Attribute options.

51905 The **restrict** keyword is added to the *pthread_attr_getstacksize()* prototype for alignment with the
51906 ISO/IEC 9899:1999 standard.

51907 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code
51908 in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack
51909 Address Attribute” option to “Thread Stack Size Attribute” option.

51910 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/87 is applied, updating the ERRORS
51911 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
51912 object.

51913 **Issue 7**

51914 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are marked only as part of
51915 the Thread Stack Size Attribute option as the Threads option is now part of the Base.

51916 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
51917 results in undefined behavior.

51918 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0265 [757] is applied.

51919 **NAME**

51920 pthread_attr_init — initialize the thread attributes object

51921 **SYNOPSIS**

51922 #include <pthread.h>

51923 int pthread_attr_init(pthread_attr_t *attr);

51924 **DESCRIPTION**

51925 Refer to *pthread_attr_destroy()*.

51926 **NAME**

51927 pthread_attr_setdetachstate — set the detachstate attribute

51928 **SYNOPSIS**

51929 #include <pthread.h>

51930 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);

51931 **DESCRIPTION**

51932 Refer to *pthread_attr_getdetachstate()*.

51933 **NAME**

51934 pthread_attr_setguardsize — set the thread guardsize attribute

51935 **SYNOPSIS**

51936 #include <pthread.h>

51937 int pthread_attr_setguardsize(pthread_attr_t *attr,
51938 size_t guardsize);

51939 **DESCRIPTION**

51940 Refer to *pthread_attr_getguardsize()*.

51941 **NAME**51942 pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)51943 **SYNOPSIS**

```
51944 TPS #include <pthread.h>
51945      int pthread_attr_setinheritsched(pthread_attr_t *attr,
51946      int inheritsched);
```

51947 **DESCRIPTION**51948 Refer to *pthread_attr_getinheritsched()*.

51949 **NAME**

51950 pthread_attr_setschedparam — set the schedparam attribute

51951 **SYNOPSIS**

51952 #include <pthread.h>

51953 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
51954 const struct sched_param *restrict param);

51955 **DESCRIPTION**

51956 Refer to [pthread_attr_getschedparam\(\)](#).

51957 **NAME**51958 pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)51959 **SYNOPSIS**

```
51960 TPS #include <pthread.h>
51961 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

51962 **DESCRIPTION**51963 Refer to *pthread_attr_getschedpolicy()*.

51964 **NAME**

51965 pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)

51966 **SYNOPSIS**

```
51967 TPS #include <pthread.h>  
51968 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

51969 **DESCRIPTION**

51970 Refer to *pthread_attr_getscope()*.

51971 **NAME**

51972 pthread_attr_setstack — set the stack attribute

51973 **SYNOPSIS**

```
51974 TSA TSS #include <pthread.h>
51975 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
51976 size_t stacksize);
```

51977 **DESCRIPTION**51978 Refer to [pthread_attr_getstack\(\)](#).

51979 **NAME**

51980 pthread_attr_setstacksize — set the stacksize attribute

51981 **SYNOPSIS**

```
51982 TSS #include <pthread.h>  
51983 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

51984 **DESCRIPTION**

51985 Refer to [pthread_attr_getstacksize\(\)](#).

51986 **NAME**

51987 pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object

51988 **SYNOPSIS**

51989 #include <pthread.h>

51990 int pthread_barrier_destroy(pthread_barrier_t *barrier);

51991 int pthread_barrier_init(pthread_barrier_t *restrict barrier,

51992 const pthread_barrierattr_t *restrict attr, unsigned count);

51993 **DESCRIPTION**

51994 The *pthread_barrier_destroy()* function shall destroy the barrier referenced by *barrier* and release
 51995 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until
 51996 the barrier is reinitialized by another call to *pthread_barrier_init()*. An implementation may use
 51997 this function to set *barrier* to an invalid value. The results are undefined if
 51998 *pthread_barrier_destroy()* is called when any thread is blocked on the barrier, or if this function is
 51999 called with an uninitialized barrier.

52000 The *pthread_barrier_init()* function shall allocate any resources required to use the barrier
 52001 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is
 52002 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of
 52003 a default barrier attributes object. The results are undefined if *pthread_barrier_init()* is called
 52004 when any thread is blocked on the barrier (that is, has not returned from the
 52005 *pthread_barrier_wait()* call). The results are undefined if a barrier is used without first being
 52006 initialized. The results are undefined if *pthread_barrier_init()* is called specifying an already
 52007 initialized barrier.

52008 The *count* argument specifies the number of threads that must call *pthread_barrier_wait()* before
 52009 any of them successfully return from the call. The value specified by *count* must be greater than
 52010 zero.

52011 If the *pthread_barrier_init()* function fails, the barrier shall not be initialized and the contents of
 52012 *barrier* are undefined.

52013 See [Section 2.9.9](#) (on page 523) for further requirements.

52014 **RETURN VALUE**

52015 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 52016 be returned to indicate the error.

52017 **ERRORS**

52018 The *pthread_barrier_init()* function shall fail if:

52019 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

52020 [EINVAL] The value specified by *count* is equal to zero.

52021 [ENOMEM] Insufficient memory exists to initialize the barrier.

52022 These functions shall not return an error code of [EINTR].

52023 **EXAMPLES**

52024 None.

52025 **APPLICATION USAGE**

52026 None.

52027 **RATIONALE**

52028 If an implementation detects that the value specified by the *barrier* argument to
52029 *pthread_barrier_destroy()* does not refer to an initialized barrier object, it is recommended that the
52030 function should fail and report an [EINVAL] error.

52031 If an implementation detects that the value specified by the *attr* argument to
52032 *pthread_barrier_init()* does not refer to an initialized barrier attributes object, it is recommended
52033 that the function should fail and report an [EINVAL] error.

52034 If an implementation detects that the value specified by the *barrier* argument to
52035 *pthread_barrier_destroy()* or *pthread_barrier_init()* refers to a barrier that is in use (for example, in
52036 a *pthread_barrier_wait()* call) by another thread, or detects that the value specified by the *barrier*
52037 argument to *pthread_barrier_init()* refers to an already initialized barrier object, it is
52038 recommended that the function should fail and report an [EBUSY] error.

52039 **FUTURE DIRECTIONS**

52040 None.

52041 **SEE ALSO**52042 [*pthread_barrier_wait\(\)*](#)52043 XBD <[pthread.h](#)>52044 **CHANGE HISTORY**

52045 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

52046 **Issue 7**

52047 The *pthread_barrier_destroy()* and *pthread_barrier_init()* functions are moved from the Barriers
52048 option to the Base.

52049 The [EINVAL] error for an uninitialized barrier object and an uninitialized barrier attributes
52050 object is removed; this condition results in undefined behavior.

52051 The [EBUSY] error for a barrier that is in use or an already initialized barrier object is removed;
52052 this condition results in undefined behavior.

52053 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0266 [972] is applied.

52054 **NAME**

52055 pthread_barrier_wait — synchronize at a barrier

52056 **SYNOPSIS**

52057 #include <pthread.h>

52058 int pthread_barrier_wait(pthread_barrier_t *barrier);

52059 **DESCRIPTION**

52060 The *pthread_barrier_wait()* function shall synchronize participating threads at the barrier
52061 referenced by *barrier*. The calling thread shall block until the required number of threads have
52062 called *pthread_barrier_wait()* specifying the barrier.

52063 When the required number of threads have called *pthread_barrier_wait()* specifying the barrier,
52064 the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified
52065 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall
52066 be reset to the state it had as a result of the most recent *pthread_barrier_init()* function that
52067 referenced it.

52068 The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in <pthread.h> and its value
52069 shall be distinct from any other value returned by *pthread_barrier_wait()*.

52070 The results are undefined if this function is called with an uninitialized barrier.

52071 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the
52072 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the
52073 required number of threads have not arrived at the barrier during the execution of the signal
52074 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until
52075 the thread in the signal handler returns from it, it is unspecified whether other threads may
52076 proceed past the barrier once they have all reached it.

52077 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to
52078 use the same processing resources from eventually making forward progress in its execution.
52079 Eligibility for processing resources shall be determined by the scheduling policy.

52080 **RETURN VALUE**

52081 Upon successful completion, the *pthread_barrier_wait()* function shall return
52082 PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the
52083 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to
52084 indicate the error.

52085 **ERRORS**

52086 This function shall not return an error code of [EINTR].

52087 **EXAMPLES**

52088 None.

52089 **APPLICATION USAGE**

52090 Applications using this function may be subject to priority inversion, as discussed in XBD
52091 [Section 3.291](#) (on page 80).

52092 **RATIONALE**

52093 If an implementation detects that the value specified by the *barrier* argument to
52094 *pthread_barrier_wait()* does not refer to an initialized barrier object, it is recommended that the
52095 function should fail and report an [EINVAL] error.

52096 **FUTURE DIRECTIONS**

52097 None.

52098 **SEE ALSO**

52099 [pthread_barrier_destroy\(\)](#)

52100 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<pthread.h>](#)

52101 **CHANGE HISTORY**

52102 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

52103 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

52104 **Issue 7**

52105 The [pthread_barrier_wait\(\)](#) function is moved from the Barriers option to the Base.

52106 The [EINVAL] error for an uninitialized barrier object is removed; this condition results in
52107 undefined behavior.

52108 NAME

52109 pthread_barrierattr_destroy, pthread_barrierattr_init — destroy and initialize the barrier
52110 attributes object

52111 SYNOPSIS

```
52112 #include <pthread.h>  
52113 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);  
52114 int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

52115 DESCRIPTION

52116 The *pthread_barrierattr_destroy()* function shall destroy a barrier attributes object. A destroyed
52117 *attr* attributes object can be reinitialized using *pthread_barrierattr_init()*; the results of otherwise
52118 referencing the object after it has been destroyed are undefined. An implementation may cause
52119 *pthread_barrierattr_destroy()* to set the object referenced by *attr* to an invalid value.

52120 The *pthread_barrierattr_init()* function shall initialize a barrier attributes object *attr* with the
52121 default value for all of the attributes defined by the implementation.

52122 If *pthread_barrierattr_init()* is called specifying an already initialized *attr* attributes object, the
52123 results are undefined.

52124 After a barrier attributes object has been used to initialize one or more barriers, any function
52125 affecting the attributes object (including destruction) shall not affect any previously initialized
52126 barrier.

52127 The behavior is undefined if the value specified by the *attr* argument to
52128 *pthread_barrierattr_destroy()* does not refer to an initialized barrier attributes object.

52129 RETURN VALUE

52130 If successful, the *pthread_barrierattr_destroy()* and *pthread_barrierattr_init()* functions shall return
52131 zero; otherwise, an error number shall be returned to indicate the error.

52132 ERRORS

52133 The *pthread_barrierattr_init()* function shall fail if:

52134 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

52135 These functions shall not return an error code of [EINTR].

52136 EXAMPLES

52137 None.

52138 APPLICATION USAGE

52139 None.

52140 RATIONALE

52141 If an implementation detects that the value specified by the *attr* argument to
52142 *pthread_barrierattr_destroy()* does not refer to an initialized barrier attributes object, it is
52143 recommended that the function should fail and report an [EINVAL] error.

52144 FUTURE DIRECTIONS

52145 None.

52146 SEE ALSO

52147 [pthread_barrierattr_getshared\(\)](#)

52148 XBD [<pthread.h>](#)

52149 CHANGE HISTORY

52150 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

52151 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

52152 Issue 7

52153 The `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions are moved from the
52154 Barriers option to the Base.

52155 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition
52156 results in undefined behavior.

52157 **NAME**

52158 pthread_barrierattr_getpshared, pthread_barrierattr_setpshared ‡ get and set the process-
 52159 shared attribute of the barrier attributes object

52160 **SYNOPSIS**

```
52161 TSH #include <pthread.h>
52162
52162 int pthread_barrierattr_getpshared(const pthread_barrierattr_t
52163     *restrict attr, int *restrict pshared);
52164 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
52165     int pshared);
```

52166 **DESCRIPTION**

52167 The *pthread_barrierattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 52168 from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared()* function shall
 52169 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

52170 The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a barrier to be
 52171 operated upon by any thread that has access to the memory where the barrier is allocated. See
 52172 [Section 2.9.9](#) (on page 523) for further requirements. The default value of the attribute shall be
 52173 PTHREAD_PROCESS_PRIVATE. Both constants PTHREAD_PROCESS_SHARED and
 52174 PTHREAD_PROCESS_PRIVATE are defined in **<pthread.h>**.

52175 Additional attributes, their default values, and the names of the associated functions to get and
 52176 set those attribute values are implementation-defined.

52177 The behavior is undefined if the value specified by the *attr* argument to
 52178 *pthread_barrierattr_getpshared()* or *pthread_barrierattr_setpshared()* does not refer to an initialized
 52179 barrier attributes object.

52180 **RETURN VALUE**

52181 If successful, the *pthread_barrierattr_getpshared()* function shall return zero and store the value of
 52182 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 52183 an error number shall be returned to indicate the error.

52184 If successful, the *pthread_barrierattr_setpshared()* function shall return zero; otherwise, an error
 52185 number shall be returned to indicate the error.

52186 **ERRORS**

52187 The *pthread_barrierattr_setpshared()* function may fail if:

52188 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal
 52189 values PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

52190 These functions shall not return an error code of [EINTR].

52191 **EXAMPLES**

52192 None.

52193 **APPLICATION USAGE**

52194 The *pthread_barrierattr_getpshared()* and *pthread_barrierattr_setpshared()* functions are part of the
 52195 Thread Process-Shared Synchronization option and need not be provided on all
 52196 implementations.

52197 **RATIONALE**

52198 If an implementation detects that the value specified by the *attr* argument to
 52199 *pthread_barrierattr_getpshared()* or *pthread_barrierattr_setpshared()* does not refer to an initialized
 52200 barrier attributes object, it is recommended that the function should fail and report an [EINVAL]

52201 error.

52202 **FUTURE DIRECTIONS**

52203 None.

52204 **SEE ALSO**

52205 [pthread_barrier_destroy\(\)](#), [pthread_barrierattr_destroy\(\)](#)

52206 XBD [<pthread.h>](#)

52207 **CHANGE HISTORY**

52208 First released in Issue 6. Derived from IEEE Std 1003.1j-2000

52209 **Issue 7**

52210 The [pthread_barrierattr_getpshared\(\)](#) and [pthread_barrierattr_setpshared\(\)](#) functions are marked
52211 only as part of the Thread Process-Shared Synchronization option as the Threads option is now
52212 part of the Base.

52213 The [EINVAL] error for an uninitialized barrier attributes object is removed; this condition
52214 results in undefined behavior.

52215 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0266 [972] and XSH/TC2-2008/0267
52216 [757] are applied.

52217 **NAME**

52218 pthread_barrierattr_init — initialize the barrier attributes object

52219 **SYNOPSIS**

52220 #include <pthread.h>

52221 int pthread_barrierattr_init(pthread_barrierattr_t *attr);

52222 **DESCRIPTION**52223 Refer to *pthread_barrierattr_destroy()*.

52224 **NAME**

52225 pthread_barrierattr_setpshared — set the process-shared attribute of the barrier attributes object

52226 **SYNOPSIS**

```
52227 TSH #include <pthread.h>
52228      int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
52229      int pshared);
```

52230 **DESCRIPTION**

52231 Refer to [pthread_barrierattr_getpshared\(\)](#).

52232 **NAME**

52233 pthread_cancel — cancel execution of a thread

52234 **SYNOPSIS**

52235 #include <pthread.h>

52236 int pthread_cancel(pthread_t thread);

52237 **DESCRIPTION**

52238 The *pthread_cancel()* function shall request that *thread* be canceled. The target thread's
52239 cancelability state and type determines when the cancellation takes effect. When the cancellation
52240 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last
52241 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called
52242 for *thread*. When the last destructor function returns, *thread* shall be terminated.

52243 The cancellation processing in the target thread shall run asynchronously with respect to the
52244 calling thread returning from *pthread_cancel()*.

52245 **RETURN VALUE**

52246 If successful, the *pthread_cancel()* function shall return zero; otherwise, an error number shall be
52247 returned to indicate the error.

52248 **ERRORS**52249 The *pthread_cancel()* function shall not return an error code of [EINTR].52250 **EXAMPLES**

52251 None.

52252 **APPLICATION USAGE**

52253 None.

52254 **RATIONALE**

52255 Two alternative functions were considered for sending the cancellation notification to a thread.
52256 One would be to define a new SIGCANCEL signal that had the cancellation semantics when
52257 delivered; the other was to define the new *pthread_cancel()* function, which would trigger the
52258 cancellation semantics.

52259 The advantage of a new signal was that so much of the delivery criteria were identical to that
52260 used when trying to deliver a signal that making cancellation notification a signal was seen as
52261 consistent. Indeed, many implementations implement cancellation using a special signal. On the
52262 other hand, there would be no signal functions that could be used with this signal except
52263 *pthread_kill()*, and the behavior of the delivered cancellation signal would be unlike any
52264 previously existing defined signal.

52265 The benefits of a special function include the recognition that this signal would be defined
52266 because of the similar delivery criteria and that this is the only common behavior between a
52267 cancellation request and a signal. In addition, the cancellation delivery mechanism does not
52268 have to be implemented as a signal. There are also strong, if not stronger, parallels with
52269 language exception mechanisms than with signals that are potentially obscured if the delivery
52270 mechanism is visibly closer to signals.

52271 In the end, it was considered that as there were so many exceptions to the use of the new signal
52272 with existing signals functions it would be misleading. A special function has resolved this
52273 problem. This function was carefully defined so that an implementation wishing to provide the
52274 cancellation functions on top of signals could do so. The special function also means that
52275 implementations are not obliged to implement cancellation with signals.

52276 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
52277 that the function should fail and report an [ESRCH] error.

52278 **FUTURE DIRECTIONS**

52279 None.

52280 **SEE ALSO**

52281 *pthread_exit()*, *pthread_cond_timedwait()*, *pthread_join()*, *pthread_setcancelstate()*

52282 XBD <pthread.h>

52283 **CHANGE HISTORY**

52284 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52285 **Issue 6**

52286 The *pthread_cancel()* function is marked as part of the Threads option.

52287 **Issue 7**

52288 The *pthread_cancel()* function is moved from the Threads option to the Base.

52289 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

52290 **NAME**

52291 pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers

52292 **SYNOPSIS**

52293 #include <pthread.h>

52294 void pthread_cleanup_pop(int *execute*);52295 void pthread_cleanup_push(void (**routine*)(void*), void **arg*);52296 **DESCRIPTION**52297 The *pthread_cleanup_pop()* function shall remove the routine at the top of the calling thread's
52298 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).52299 The *pthread_cleanup_push()* function shall push the specified cancellation cleanup handler *routine*
52300 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be
52301 popped from the cancellation cleanup stack and invoked with the argument *arg* when:52302 The thread exits (that is, calls *pthread_exit()*).

52303 The thread acts upon a cancellation request.

52304 The thread calls *pthread_cleanup_pop()* with a non-zero *execute* argument.52305 It is unspecified whether *pthread_cleanup_push()* and *pthread_cleanup_pop()* are macros or
52306 functions. If a macro definition is suppressed in order to access an actual function, or a program
52307 defines an external identifier with any of these names, the behavior is undefined. The
52308 application shall ensure that they appear as statements, and in pairs within the same lexical
52309 scope (that is, the *pthread_cleanup_push()* macro may be thought to expand to a token list whose
52310 first token is '{' with *pthread_cleanup_pop()* expanding to a token list whose last token is the
52311 corresponding '}').52312 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to
52313 *pthread_cleanup_push()* or *pthread_cleanup_pop()* made without the matching call since the jump
52314 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation
52315 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation
52316 cleanup handler.52317 The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block
52318 described by a pair of *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions calls is
52319 undefined.52320 **RETURN VALUE**52321 The *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions shall not return a value.52322 **ERRORS**

52323 No errors are defined.

52324 These functions shall not return an error code of [EINTR].

52325 **EXAMPLES**52326 The following is an example using thread primitives to implement a cancelable, writers-priority
52327 read-write lock:52328 typedef struct {
52329 pthread_mutex_t lock;
52330 pthread_cond_t rcond,
52331 wcond;
52332 int lock_count; /* < 0 .. Held by writer. */
52333 /* > 0 .. Held by lock_count readers. */
52334 /* = 0 .. Held by nobody. */

```
52335     int waiting_writers; /* Count of waiting writers. */
52336 } rwlock;

52337 void
52338 waiting_reader_cleanup(void *arg)
52339 {
52340     rwlock *l;

52341     l = (rwlock *) arg;
52342     pthread_mutex_unlock(&l->lock);
52343 }

52344 void
52345 lock_for_read(rwlock *l)
52346 {
52347     pthread_mutex_lock(&l->lock);
52348     pthread_cleanup_push(waiting_reader_cleanup, l);
52349     while ((l->lock_count < 0) || (l->waiting_writers != 0))
52350         pthread_cond_wait(&l->rcond, &l->lock);
52351     l->lock_count++;
52352     /*
52353      * Note the pthread_cleanup_pop executes
52354      * waiting_reader_cleanup.
52355      */
52356     pthread_cleanup_pop(1);
52357 }

52358 void
52359 release_read_lock(rwlock *l)
52360 {
52361     pthread_mutex_lock(&l->lock);
52362     if (--l->lock_count == 0)
52363         pthread_cond_signal(&l->wcond);
52364     pthread_mutex_unlock(&l->lock);
52365 }

52366 void
52367 waiting_writer_cleanup(void *arg)
52368 {
52369     rwlock *l;

52370     l = (rwlock *) arg;
52371     if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
52372         /*
52373          * This only happens if we have been canceled. If the
52374          * lock is not held by a writer, there may be readers who
52375          * were blocked because waiting_writers was positive; they
52376          * can now be unblocked.
52377          */
52378         pthread_cond_broadcast(&l->rcond);
52379     }
52380     pthread_mutex_unlock(&l->lock);
52381 }

52382 void
```

```
52383     lock_for_write(rwlock *l)
52384     {
52385         pthread_mutex_lock(&l->lock);
52386         l->waiting_writers++;
52387         pthread_cleanup_push(waiting_writer_cleanup, l);
52388         while (l->lock_count != 0)
52389             pthread_cond_wait(&l->wcond, &l->lock);
52390         l->lock_count = -1;
52391         /*
52392          * Note the pthread_cleanup_pop executes
52393          * waiting_writer_cleanup.
52394          */
52395         pthread_cleanup_pop(1);
52396     }
52397
52398 void
52399 release_write_lock(rwlock *l)
52400 {
52401     pthread_mutex_lock(&l->lock);
52402     l->lock_count = 0;
52403     if (l->waiting_writers == 0)
52404         pthread_cond_broadcast(&l->rcond);
52405     else
52406         pthread_cond_signal(&l->wcond);
52407     pthread_mutex_unlock(&l->lock);
52408 }
52409 /*
52410  * This function is called to initialize the read/write lock.
52411  */
52412 void
52413 initialize_rwlock(rwlock *l)
52414 {
52415     pthread_mutex_init(&l->lock, pthread_mutexattr_default);
52416     pthread_cond_init(&l->wcond, pthread_condattr_default);
52417     pthread_cond_init(&l->rcond, pthread_condattr_default);
52418     l->lock_count = 0;
52419     l->waiting_writers = 0;
52420 }
52421
52422 void
52423 reader_thread()
52424 {
52425     lock_for_read(&lock);
52426     pthread_cleanup_push(release_read_lock, &lock);
52427     /*
52428      * Thread has read lock.
52429      */
52430     pthread_cleanup_pop(1);
52431 }
52432
52433 void
52434 writer_thread()
52435 {
52436     lock_for_write(&lock);
52437     pthread_cleanup_push(release_write_lock, &lock);
```

```

52433     /*
52434     * Thread has write lock.
52435     */
52436     pthread_cleanup_pop(1);
52437 }

```

52438 APPLICATION USAGE

52439 The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push()* and
52440 *pthread_cleanup_pop()*, can be thought of as left and right-parentheses. They always need to be
52441 matched.

52442 RATIONALE

52443 The restriction that the two routines that push and pop cancellation cleanup handlers,
52444 *pthread_cleanup_push()* and *pthread_cleanup_pop()*, have to appear in the same lexical scope
52445 allows for efficient macro or compiler implementations and efficient storage management. A
52446 sample implementation of these routines as macros might look like this:

```

52447 #define pthread_cleanup_push(rtn,arg) { \
52448     struct _pthread_handler_rec __cleanup_handler, **__head; \
52449     __cleanup_handler.rtn = rtn; \
52450     __cleanup_handler.arg = arg; \
52451     (void) pthread_getspecific(_pthread_handler_key, &__head); \
52452     __cleanup_handler.next = *__head; \
52453     *__head = &__cleanup_handler;
52454
52455 #define pthread_cleanup_pop(ex) \
52456     *_head = __cleanup_handler.next; \
52457     if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
52458 }

```

52458 A more ambitious implementation of these routines might do even better by allowing the
52459 compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

52460 This volume of POSIX.1-2017 currently leaves unspecified the effect of calling *longjmp()* from a
52461 signal handler executing in a POSIX System Interfaces function. If an implementation wants to
52462 allow this and give the programmer reasonable behavior, the *longjmp()* function has to call all
52463 cancellation cleanup handlers that have been pushed but not popped since the time *setjmp()* was
52464 called.

52465 Consider a multi-threaded function called by a thread that uses signals. If a signal were
52466 delivered to a signal handler during the operation of *qsort()* and that handler were to call
52467 *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads
52468 created by the *qsort()* function would not be canceled. Instead, they would continue to execute
52469 and write into the argument array even though the array might have been popped off the stack.

52470 Note that the specified cleanup handling mechanism is especially tied to the C language and,
52471 while the requirement for a uniform mechanism for expressing cleanup is language-
52472 independent, the mechanism used in other languages may be quite different. In addition, this
52473 mechanism is really only necessary due to the lack of a real exception mechanism in the C
52474 language, which would be the ideal solution.

52475 There is no notion of a cancellation cleanup-safe function. If an application has no cancellation
52476 points in its signal handlers, blocks any signal whose handler may have cancellation points
52477 while calling async-unsafe functions, or disables cancellation while calling async-unsafe
52478 functions, all functions may be safely called from cancellation cleanup routines.

52479 **FUTURE DIRECTIONS**

52480 None.

52481 **SEE ALSO**52482 *pthread_cancel()*, *pthread_setcancelstate()*

52483 XBD <pthread.h>

52484 **CHANGE HISTORY**

52485 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52486 **Issue 6**52487 The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are marked as part of the
52488 Threads option.

52489 The APPLICATION USAGE section is added.

52490 The normative text is updated to avoid use of the term “must” for application requirements.

52491 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/88 is applied, updating the
52492 DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the
52493 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions.52494 **Issue 7**52495 The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are moved from the Threads
52496 option to the Base.

52497 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0454 [229] is applied.

52498 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0268 [624] is applied.

52499 **NAME**

52500 pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

52501 **SYNOPSIS**

52502 #include <pthread.h>

52503 int pthread_cond_broadcast(pthread_cond_t *cond);

52504 int pthread_cond_signal(pthread_cond_t *cond);

52505 **DESCRIPTION**

52506 These functions shall unblock threads blocked on a condition variable.

52507 The *pthread_cond_broadcast()* function shall unblock all threads currently blocked on the
52508 specified condition variable *cond*.

52509 The *pthread_cond_signal()* function shall unblock at least one of the threads that are blocked on
52510 the specified condition variable *cond* (if any threads are blocked on *cond*).

52511 If more than one thread is blocked on a condition variable, the scheduling policy shall determine
52512 the order in which threads are unblocked. When each thread unblocked as a result of a
52513 *pthread_cond_broadcast()* or *pthread_cond_signal()* returns from its call to *pthread_cond_wait()* or
52514 *pthread_cond_timedwait()*, the thread shall own the mutex with which it called
52515 *pthread_cond_wait()* or *pthread_cond_timedwait()*. The thread(s) that are unblocked shall contend
52516 for the mutex according to the scheduling policy (if applicable), and as if each had called
52517 *pthread_mutex_lock()*.

52518 The *pthread_cond_broadcast()* or *pthread_cond_signal()* functions may be called by a thread
52519 whether or not it currently owns the mutex that threads calling *pthread_cond_wait()* or
52520 *pthread_cond_timedwait()* have associated with the condition variable during their waits;
52521 however, if predictable scheduling behavior is required, then that mutex shall be locked by the
52522 thread calling *pthread_cond_broadcast()* or *pthread_cond_signal()*.

52523 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall have no effect if there are
52524 no threads currently blocked on *cond*.

52525 The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_broadcast()*
52526 or *pthread_cond_signal()* does not refer to an initialized condition variable.

52527 **RETURN VALUE**

52528 If successful, the *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall return zero;
52529 otherwise, an error number shall be returned to indicate the error.

52530 **ERRORS**

52531 These functions shall not return an error code of [EINTR].

52532 **EXAMPLES**

52533 None.

52534 **APPLICATION USAGE**

52535 The *pthread_cond_broadcast()* function is used whenever the shared-variable state has been
52536 changed in a way that more than one thread can proceed with its task. Consider a single
52537 producer/multiple consumer problem, where the producer can insert multiple items on a list
52538 that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast()*
52539 function, the producer would notify all consumers that might be waiting, and thereby the
52540 application would receive more throughput on a multi-processor. In addition,
52541 *pthread_cond_broadcast()* makes it easier to implement a read-write lock. The
52542 *pthread_cond_broadcast()* function is needed in order to wake up all waiting readers when a
52543 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function
52544 to notify all clients of an impending transaction commit.

52545 It is not safe to use the *pthread_cond_signal()* function in a signal handler that is invoked
 52546 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean
 52547 *pthread_cond_wait()* that could not be efficiently eliminated.

52548 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling
 52549 from code running in a signal handler.

52550 RATIONALE

52551 If an implementation detects that the value specified by the *cond* argument to
 52552 *pthread_cond_broadcast()* or *pthread_cond_signal()* does not refer to an initialized condition
 52553 variable, it is recommended that the function should fail and report an [EINVAL] error.

52554 Multiple Awakenings by Condition Signal

52555 On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal()* to
 52556 avoid the unblocking of more than one thread blocked on a condition variable. For example,
 52557 consider the following partial implementation of *pthread_cond_wait()* and *pthread_cond_signal()*,
 52558 executed by two threads in the order given. One thread is trying to wait on the condition
 52559 variable, another is concurrently executing *pthread_cond_signal()*, while a third thread is already
 52560 waiting.

```
52561 pthread_cond_wait(mutex, cond):
52562     value = cond->value; /* 1 */
52563     pthread_mutex_unlock(mutex); /* 2 */
52564     pthread_mutex_lock(cond->mutex); /* 10 */
52565     if (value == cond->value) { /* 11 */
52566         me->next_cond = cond->waiter;
52567         cond->waiter = me;
52568         pthread_mutex_unlock(cond->mutex);
52569         unable_to_run(me);
52570     } else
52571         pthread_mutex_unlock(cond->mutex); /* 12 */
52572     pthread_mutex_lock(mutex); /* 13 */

52573 pthread_cond_signal(cond):
52574     pthread_mutex_lock(cond->mutex); /* 3 */
52575     cond->value++; /* 4 */
52576     if (cond->waiter) { /* 5 */
52577         sleeper = cond->waiter; /* 6 */
52578         cond->waiter = sleeper->next_cond; /* 7 */
52579         able_to_run(sleeper); /* 8 */
52580     }
52581     pthread_mutex_unlock(cond->mutex); /* 9 */
```

52582 The effect is that more than one thread can return from its call to *pthread_cond_wait()* or
 52583 *pthread_cond_timedwait()* as a result of one call to *pthread_cond_signal()*. This effect is called
 52584 “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that
 52585 are so awakened is finite; for example, the next thread to call *pthread_cond_wait()* after the
 52586 sequence of events above blocks.

52587 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs
 52588 only rarely is unacceptable, especially given that one has to check the predicate associated with a
 52589 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of
 52590 concurrency in this basic building block for all higher-level synchronization operations.

52591 An added benefit of allowing spurious wakeups is that applications are forced to code a

52592 predicate-testing-loop around the condition wait. This also makes the application tolerate
52593 superfluous condition broadcasts or signals on the same condition variable that may be coded in
52594 some other part of the application. The resulting applications are thus more robust. Therefore,
52595 POSIX.1-2017 explicitly documents that spurious wakeups may occur.

52596 FUTURE DIRECTIONS

52597 None.

52598 SEE ALSO

52599 [*pthread_cond_destroy\(\)*](#), [*pthread_cond_timedwait\(\)*](#)

52600 XBD [Section 4.12](#) (on page 111), [**<pthread.h>**](#)

52601 CHANGE HISTORY

52602 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52603 Issue 6

52604 The [*pthread_cond_broadcast\(\)*](#) and [*pthread_cond_signal\(\)*](#) functions are marked as part of the
52605 Threads option.

52606 The APPLICATION USAGE section is added.

52607 Issue 7

52608 The [*pthread_cond_broadcast\(\)*](#) and [*pthread_cond_signal\(\)*](#) functions are moved from the Threads
52609 option to the Base.

52610 The [EINVAL] error for an uninitialized condition variable is removed; this condition results in
52611 undefined behavior.

52612 **NAME**

52613 pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

52614 **SYNOPSIS**

```
52615 #include <pthread.h>
52616 int pthread_cond_destroy(pthread_cond_t *cond);
52617 int pthread_cond_init(pthread_cond_t *restrict cond,
52618     const pthread_condattr_t *restrict attr);
52619 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

52620 **DESCRIPTION**

52621 The *pthread_cond_destroy()* function shall destroy the given condition variable specified by *cond*;
 52622 the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy()*
 52623 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can
 52624 be reinitialized using *pthread_cond_init()*; the results of otherwise referencing the object after it
 52625 has been destroyed are undefined.

52626 It shall be safe to destroy an initialized condition variable upon which no threads are currently
 52627 blocked. Attempting to destroy a condition variable upon which other threads are currently
 52628 blocked results in undefined behavior.

52629 The *pthread_cond_init()* function shall initialize the condition variable referenced by *cond* with
 52630 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be
 52631 used; the effect is the same as passing the address of a default condition variable attributes
 52632 object. Upon successful initialization, the state of the condition variable shall become initialized.

52633 See [Section 2.9.9](#) (on page 523) for further requirements.

52634 Attempting to initialize an already initialized condition variable results in undefined behavior.

52635 In cases where default condition variable attributes are appropriate, the macro
 52636 PTHREAD_COND_INITIALIZER can be used to initialize condition variables. The effect shall
 52637 be equivalent to dynamic initialization by a call to *pthread_cond_init()* with parameter *attr*
 52638 specified as NULL, except that no error checks are performed.

52639 The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_destroy()*
 52640 does not refer to an initialized condition variable.

52641 The behavior is undefined if the value specified by the *attr* argument to *pthread_cond_init()* does
 52642 not refer to an initialized condition variable attributes object.

52643 **RETURN VALUE**

52644 If successful, the *pthread_cond_destroy()* and *pthread_cond_init()* functions shall return zero;
 52645 otherwise, an error number shall be returned to indicate the error.

52646 **ERRORS**

52647 The *pthread_cond_init()* function shall fail if:

52648 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 52649 another condition variable.

52650 [ENOMEM] Insufficient memory exists to initialize the condition variable.

52651 These functions shall not return an error code of [EINTR].

52652 **EXAMPLES**

52653 A condition variable can be destroyed immediately after all the threads that are blocked on it are
52654 awakened. For example, consider the following code:

```
52655 struct list {
52656     pthread_mutex_t lm;
52657     ...
52658 }

52659 struct elt {
52660     key k;
52661     int busy;
52662     pthread_cond_t notbusy;
52663     ...
52664 }

52665 /* Find a list element and reserve it. */
52666 struct elt *
52667 list_find(struct list *lp, key k)
52668 {
52669     struct elt *ep;

52670     pthread_mutex_lock(&lp->lm);
52671     while ((ep = find_elt(l, k) != NULL) && ep->busy)
52672         pthread_cond_wait(&ep->notbusy, &lp->lm);
52673     if (ep != NULL)
52674         ep->busy = 1;
52675     pthread_mutex_unlock(&lp->lm);
52676     return(ep);
52677 }

52678 delete_elt(struct list *lp, struct elt *ep)
52679 {
52680     pthread_mutex_lock(&lp->lm);
52681     assert(ep->busy);
52682     ... remove ep from list ...
52683     ep->busy = 0; /* Paranoid. */
52684     (A) pthread_cond_broadcast(&ep->notbusy);
52685     pthread_mutex_unlock(&lp->lm);
52686     (B) pthread_cond_destroy(&ep->notbusy);
52687     free(ep);
52688 }
```

52689 In this example, the condition variable and its list element may be freed (line B) immediately
52690 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that
52691 no other thread can touch the element to be deleted.

52692 **APPLICATION USAGE**

52693 None.

52694 **RATIONALE**

52695 If an implementation detects that the value specified by the *cond* argument to
52696 *pthread_cond_destroy()* does not refer to an initialized condition variable, it is recommended that
52697 the function should fail and report an [EINVAL] error.

52698 If an implementation detects that the value specified by the *cond* argument to

52699 *pthread_cond_destroy()* or *pthread_cond_init()* refers to a condition variable that is in use (for
 52700 example, in a *pthread_cond_wait()* call) by another thread, or detects that the value specified by
 52701 the *cond* argument to *pthread_cond_init()* refers to an already initialized condition variable, it is
 52702 recommended that the function should fail and report an [EBUSY] error.

52703 If an implementation detects that the value specified by the *attr* argument to *pthread_cond_init()*
 52704 does not refer to an initialized condition variable attributes object, it is recommended that the
 52705 function should fail and report an [EINVAL] error.

52706 See also *pthread_mutex_destroy()*.

52707 FUTURE DIRECTIONS

52708 None.

52709 SEE ALSO

52710 *pthread_cond_broadcast()*, *pthread_cond_timedwait()*, *pthread_mutex_destroy()*

52711 XBD <pthread.h>

52712 CHANGE HISTORY

52713 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52714 Issue 6

52715 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are marked as part of the Threads
 52716 option.

52717 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

52718 The **restrict** keyword is added to the *pthread_cond_init()* prototype for alignment with the
 52719 ISO/IEC 9899:1999 standard.

52720 Issue 7

52721 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are moved from the Threads option
 52722 to the Base.

52723 The [EINVAL] error for an uninitialized condition variable and an uninitialized condition
 52724 variable attributes object is removed; this condition results in undefined behavior.

52725 The [EBUSY] error for a condition variable already in use or an already initialized condition
 52726 variable is removed; this condition results in undefined behavior.

52727 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0455 [70] is applied.

52728 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0269 [972] and XSH/TC2-2008/0270
 52729 [910] are applied.

52730 **NAME**

52731 pthread_cond_signal — signal a condition

52732 **SYNOPSIS**

52733 #include <pthread.h>

52734 int pthread_cond_signal(pthread_cond_t *cond);

52735 **DESCRIPTION**

52736 Refer to *pthread_cond_broadcast()*.

52737 **NAME**

52738 pthread_cond_timedwait, pthread_cond_wait — wait on a condition

52739 **SYNOPSIS**

```
52740 #include <pthread.h>
52741 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
52742     pthread_mutex_t *restrict mutex,
52743     const struct timespec *restrict abstime);
52744 int pthread_cond_wait(pthread_cond_t *restrict cond,
52745     pthread_mutex_t *restrict mutex);
```

52746 **DESCRIPTION**

52747 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions shall block on a condition
 52748 variable. The application shall ensure that these functions are called with *mutex* locked by the
 52749 calling thread; otherwise, an error (for PTHREAD_MUTEX_ERRORCHECK and robust
 52750 mutexes) or undefined behavior (for other mutexes) results.

52751 These functions atomically release *mutex* and cause the calling thread to block on the condition
 52752 variable *cond*; atomically here means “atomically with respect to access by another thread to the
 52753 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex
 52754 after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast()*
 52755 or *pthread_cond_signal()* in that thread shall behave as if it were issued after the about-to-block
 52756 thread has blocked.

52757 Upon successful return, the mutex shall have been locked and shall be owned by the calling
 52758 thread. If *mutex* is a robust mutex where an owner terminated while holding the lock and the
 52759 state is recoverable, the mutex shall be acquired even though the function returns an error code.

52760 When using condition variables there is always a Boolean predicate involving shared variables
 52761 associated with each condition wait that is true if the thread should proceed. Spurious wakeups
 52762 from the *pthread_cond_timedwait()* or *pthread_cond_wait()* functions may occur. Since the return
 52763 from *pthread_cond_timedwait()* or *pthread_cond_wait()* does not imply anything about the value of
 52764 this predicate, the predicate should be re-evaluated upon such return.

52765 When a thread waits on a condition variable, having specified a particular mutex to either the
 52766 *pthread_cond_timedwait()* or the *pthread_cond_wait()* operation, a dynamic binding is formed
 52767 between that mutex and condition variable that remains in effect as long as at least one thread is
 52768 blocked on the condition variable. During this time, the effect of an attempt by any thread to
 52769 wait on that condition variable using a different mutex is undefined. Once all waiting threads
 52770 have been unblocked (as by the *pthread_cond_broadcast()* operation), the next wait operation on
 52771 that condition variable shall form a new dynamic binding with the mutex specified by that wait
 52772 operation. Even though the dynamic binding between condition variable and mutex may be
 52773 removed or replaced between the time a thread is unblocked from a wait on the condition
 52774 variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked
 52775 thread shall always re-acquire the mutex specified in the condition wait operation call from
 52776 which it is returning.

52777 A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a
 52778 thread is set to PTHREAD_CANCEL_DEFERRED, a side-effect of acting upon a cancellation
 52779 request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first
 52780 cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up
 52781 to the point of returning from the call to *pthread_cond_timedwait()* or *pthread_cond_wait()*, but at
 52782 that point notices the cancellation request and instead of returning to the caller of
 52783 *pthread_cond_timedwait()* or *pthread_cond_wait()*, starts the thread cancellation activities, which
 52784 includes calling cancellation cleanup handlers.

52785 A thread that has been unblocked because it has been canceled while blocked in a call to
 52786 *pthread_cond_timedwait()* or *pthread_cond_wait()* shall not consume any condition signal that may
 52787 be directed concurrently at the condition variable if there are other threads blocked on the
 52788 condition variable.

52789 The *pthread_cond_timedwait()* function shall be equivalent to *pthread_cond_wait()*, except that an
 52790 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or
 52791 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time
 52792 specified by *abstime* has already been passed at the time of the call. When such timeouts occur,
 52793 *pthread_cond_timedwait()* shall nonetheless release and re-acquire the mutex referenced by *mutex*,
 52794 and may consume a condition signal directed concurrently at the condition variable.

52795 The condition variable shall have a clock attribute which specifies the clock that shall be used to
 52796 measure the time specified by the *abstime* argument. The *pthread_cond_timedwait()* function is
 52797 also a cancellation point.

52798 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal
 52799 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it
 52800 shall return zero due to spurious wakeup.

52801 The behavior is undefined if the value specified by the *cond* or *mutex* argument to these
 52802 functions does not refer to an initialized condition variable or an initialized mutex object,
 52803 respectively.

52804 RETURN VALUE

52805 Except for [ETIMEDOUT], [ENOTRECOVERABLE], and [EOWNERDEAD], all these error
 52806 checks shall act as if they were performed immediately at the beginning of processing for the
 52807 function and shall cause an error return, in effect, prior to modifying the state of the mutex
 52808 specified by *mutex* or the condition variable specified by *cond*.

52809 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall
 52810 be returned to indicate the error.

52811 ERRORS

52812 These functions shall fail if:

52813 [ENOTRECOVERABLE]

52814 The state protected by the mutex is not recoverable.

52815 [EOWNERDEAD]

52816 The mutex is a robust mutex and the process containing the previous owning
 52817 thread terminated while holding the mutex lock. The mutex lock shall be
 52818 acquired by the calling thread and it is up to the new owner to make the state
 52819 consistent.

52820 [EPERM]

52821 The mutex type is PTHREAD_MUTEX_ERRORCHECK or the mutex is a
 robust mutex, and the current thread does not own the mutex.

52822 The *pthread_cond_timedwait()* function shall fail if:

52823 [ETIMEDOUT] The time specified by *abstime* to *pthread_cond_timedwait()* has passed.

52824 [EINVAL] The *abstime* argument specified a nanosecond value less than zero or greater
 52825 than or equal to 1000 million.

52826 These functions may fail if:

52827 [EOWNERDEAD]

52828 The mutex is a robust mutex and the previous owning thread terminated
52829 while holding the mutex lock. The mutex lock shall be acquired by the calling
52830 thread and it is up to the new owner to make the state consistent.

52831 These functions shall not return an error code of [EINTR].

52832 EXAMPLES

52833 None.

52834 APPLICATION USAGE

52835 Applications that have assumed that non-zero return values are errors will need updating for
52836 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting
52837 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error
52838 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If
52839 an application is supposed to work with normal and robust mutexes, it should check all return
52840 values for error conditions and if necessary take appropriate action.

52841 RATIONALE

52842 If an implementation detects that the value specified by the *cond* argument to
52843 *pthread_cond_timedwait()* or *pthread_cond_wait()* does not refer to an initialized condition
52844 variable, or detects that the value specified by the *mutex* argument to *pthread_cond_timedwait()* or
52845 *pthread_cond_wait()* does not refer to an initialized mutex object, it is recommended that the
52846 function should fail and report an [EINVAL] error.

52847 Condition Wait Semantics

52848 It is important to note that when *pthread_cond_wait()* and *pthread_cond_timedwait()* return
52849 without error, the associated predicate may still be false. Similarly, when
52850 *pthread_cond_timedwait()* returns with the timeout error, the associated predicate may be true
52851 due to an unavoidable race between the expiration of the timeout and the predicate state change.

52852 The application needs to recheck the predicate on any return because it cannot be sure there is
52853 another thread waiting on the thread to handle the signal, and if there is not then the signal is
52854 lost. The burden is on the application to check the predicate.

52855 Some implementations, particularly on a multi-processor, may sometimes cause multiple
52856 threads to wake up when the condition variable is signaled simultaneously on different
52857 processors.

52858 In general, whenever a condition wait returns, the thread has to re-evaluate the predicate
52859 associated with the condition wait to determine whether it can safely proceed, should wait
52860 again, or should declare a timeout. A return from the wait does not imply that the associated
52861 predicate is either true or false.

52862 It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop”
52863 that checks the predicate.

52864 **Timed Wait Semantics**

52865 An absolute time measure was chosen for specifying the timeout parameter for two reasons.
52866 First, a relative time measure can be easily implemented on top of a function that specifies
52867 absolute time, but there is a race condition associated with specifying an absolute timeout on top
52868 of a function that specifies relative timeouts. For example, assume that `clock_gettime()` returns
52869 the current time and `cond_relative_timed_wait()` uses relative timeouts:

```
52870 clock_gettime(CLOCK_REALTIME, &now)  
52871 reltime = sleep_til_this_absolute_time -now;  
52872 cond_relative_timed_wait(c, m, &reltime);
```

52873 If the thread is preempted between the first statement and the last statement, the thread blocks
52874 for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout
52875 also need not be recomputed if it is used multiple times in a loop, such as that enclosing a
52876 condition wait.

52877 For cases when the system clock is advanced discontinuously by an operator, it is expected that
52878 implementations process any timed wait expiring at an intervening time as if that time had
52879 actually occurred.

52880 **Cancellation and Condition Wait**

52881 A condition wait, whether timed or not, is a cancellation point. That is, the functions
52882 `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent)
52883 cancellation request is noticed. The reason for this is that an indefinite wait is possible at these
52884 points—whatever event is being waited for, even if the program is totally correct, might never
52885 occur; for example, some input data being awaited might never be sent. By making condition
52886 wait a cancellation point, the thread can be canceled and perform its cancellation cleanup
52887 handler even though it may be stuck in some indefinite wait.

52888 A side-effect of acting on a cancellation request while a thread is blocked on a condition variable
52889 is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in
52890 order to ensure that the cancellation cleanup handler is executed in the same state as the critical
52891 code that lies both before and after the call to the condition wait function. This rule is also
52892 required when interfacing to POSIX threads from languages, such as Ada or C++, which may
52893 choose to map cancellation onto a language exception; this rule ensures that each exception
52894 handler guarding a critical section can always safely depend upon the fact that the associated
52895 mutex has already been locked regardless of exactly where within the critical section the
52896 exception was raised. Without this rule, there would not be a uniform rule that exception
52897 handlers could follow regarding the lock, and so coding would become very cumbersome.

52898 Therefore, since *some* statement has to be made regarding the state of the lock when a
52899 cancellation is delivered during a wait, a definition has been chosen that makes application
52900 coding most convenient and error free.

52901 When acting on a cancellation request while a thread is blocked on a condition variable, the
52902 implementation is required to ensure that the thread does not consume any condition signals
52903 directed at that condition variable if there are any other threads waiting on that condition
52904 variable. This rule is specified in order to avoid deadlock conditions that could occur if these
52905 two independent requests (one acting on a thread and the other acting on the condition variable)
52906 were not processed independently.

52907 Performance of Mutexes and Condition Variables

52908 Mutexes are expected to be locked only for a few instructions. This practice is almost
52909 automatically enforced by the desire of programmers to avoid long serial regions of execution
52910 (which would reduce total effective parallelism).

52911 When using mutexes and condition variables, one tries to ensure that the usual case is to lock the
52912 mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a
52913 relatively rare situation. For example, when implementing a read-write lock, code that acquires a
52914 read-lock typically needs only to increment the count of readers (under mutual-exclusion) and
52915 return. The calling thread would actually wait on the condition variable only when there is
52916 already an active writer. So the efficiency of a synchronization operation is bounded by the cost
52917 of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context
52918 switch.

52919 This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be
52920 at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable
52921 is important. The cost of waiting on a condition variable should be little more than the minimal
52922 cost for a context switch plus the time to unlock and lock the mutex.

52923 Features of Mutexes and Condition Variables

52924 It had been suggested that the mutex acquisition and release be decoupled from condition wait.
52925 This was rejected because it is the combined nature of the operation that, in fact, facilitates
52926 realtime implementations. Those implementations can atomically move a high-priority thread
52927 between the condition variable and the mutex in a manner that is transparent to the caller. This
52928 can prevent extra context switches and provide more deterministic acquisition of a mutex when
52929 the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the
52930 scheduling discipline. Furthermore, the current condition wait operation matches existing
52931 practice.

52932 Scheduling Behavior of Mutexes and Condition Variables

52933 Synchronization primitives that attempt to interfere with scheduling policy by specifying an
52934 ordering rule are considered undesirable. Threads waiting on mutexes and condition variables
52935 are selected to proceed in an order dependent upon the scheduling policy rather than in some
52936 fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which
52937 thread(s) are awakened and allowed to proceed.

52938 Timed Condition Wait

52939 The `pthread_cond_timedwait()` function allows an application to give up waiting for a particular
52940 condition after a given amount of time. An example of its use follows:

```
52941 (void) pthread_mutex_lock(&t.mn);
52942     t.waiters++;
52943     clock_gettime(CLOCK_REALTIME, &ts);
52944     ts.tv_sec += 5;
52945     rc = 0;
52946     while (! mypredicate(&t) && rc == 0)
52947         rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
52948     t.waiters--;
52949     if (rc == 0 || mypredicate(&t))
52950         setmystate(&t);
52951 (void) pthread_mutex_unlock(&t.mn);
```

52952 By making the timeout parameter absolute, it does not need to be recomputed each time the
 52953 program checks its blocking predicate. If the timeout was relative, it would have to be
 52954 recomputed before each call. This would be especially difficult since such code would need to
 52955 take into account the possibility of extra wakeups that result from extra broadcasts or signals on
 52956 the condition variable that occur before either the predicate is true or the timeout is due.

52957 FUTURE DIRECTIONS

52958 None.

52959 SEE ALSO

52960 [pthread_cond_broadcast\(\)](#)

52961 XBD Section 4.12 (on page 111), [<pthread.h>](#)

52962 CHANGE HISTORY

52963 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52964 Issue 6

52965 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions are marked as part of the
 52966 Threads option.

52967 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the
 52968 *pthread_cond_wait()* function.

52969 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics
 52970 for the Clock Selection option.

52971 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC
 52972 Interpretation 1003.1c #28.

52973 The **restrict** keyword is added to the *pthread_cond_timedwait()* and *pthread_cond_wait()*
 52974 prototypes for alignment with the ISO/IEC 9899:1999 standard.

52975 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/89 is applied, updating the
 52976 DESCRIPTION for consistency with the *pthread_cond_destroy()* function that states it is safe to
 52977 destroy an initialized condition variable upon which no threads are currently blocked.

52978 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/90 is applied, updating words in the
 52979 DESCRIPTION from "the cancelability enable state" to "the cancelability type".

52980 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/91 is applied, updating the ERRORS
 52981 section to remove the error case related to *abstime* from the *pthread_cond_wait()* function, and to
 52982 make the error case related to *abstime* mandatory for *pthread_cond_timedwait()* for consistency
 52983 with other functions.

52984 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/92 is applied, adding a new paragraph to
 52985 the RATIONALE section stating that an application should check the predicate on any return
 52986 from this function.

52987 Issue 7

52988 SD5-XSH-ERN-44 is applied, changing the definition of the "shall fail" case of the [EINVAL]
 52989 error.

52990 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

52991 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions are moved from the Threads
 52992 option to the Base.

52993 The [EINVAL] error for an uninitialized condition variable or uninitialized mutex object is
 52994 removed; this condition results in undefined behavior"

- 52995 The [EPERM] error is revised and moved to the “shall fail” list of error conditions for the
52996 *pthread_cond_timedwait()* function.
- 52997 The DESCRIPTION is updated to clarify the behavior when *mutex* is a robust mutex.
- 52998 The ERRORS section is updated to include “shall fail” cases for
52999 PTHREAD_MUTEX_ERRORCHECK mutexes.
- 53000 The DESCRIPTION is rewritten to clarify that undefined behavior occurs only for mutexes
53001 where the [EPERM] error is not mandated.
- 53002 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0456 [91,286,437] and
53003 XSH/TC1-2008/0457 [239] are applied.
- 53004 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0271 [749] is applied.

53005 **NAME**

53006 pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable
53007 attributes object

53008 **SYNOPSIS**

```
53009 #include <pthread.h>
53010 int pthread_condattr_destroy(pthread_condattr_t *attr);
53011 int pthread_condattr_init(pthread_condattr_t *attr);
```

53012 **DESCRIPTION**

53013 The *pthread_condattr_destroy()* function shall destroy a condition variable attributes object; the
53014 object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy()*
53015 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be
53016 reinitialized using *pthread_condattr_init()*; the results of otherwise referencing the object after it
53017 has been destroyed are undefined.

53018 The *pthread_condattr_init()* function shall initialize a condition variable attributes object *attr* with
53019 the default value for all of the attributes defined by the implementation.

53020 Results are undefined if *pthread_condattr_init()* is called specifying an already initialized *attr*
53021 attributes object.

53022 After a condition variable attributes object has been used to initialize one or more condition
53023 variables, any function affecting the attributes object (including destruction) shall not affect any
53024 previously initialized condition variables.

53025 This volume of POSIX.1-2017 requires two attributes, the *clock* attribute and the *process-shared*
53026 attribute.

53027 Additional attributes, their default values, and the names of the associated functions to get and
53028 set those attribute values are implementation-defined.

53029 The behavior is undefined if the value specified by the *attr* argument to
53030 *pthread_condattr_destroy()* does not refer to an initialized condition variable attributes object.

53031 **RETURN VALUE**

53032 If successful, the *pthread_condattr_destroy()* and *pthread_condattr_init()* functions shall return
53033 zero; otherwise, an error number shall be returned to indicate the error.

53034 **ERRORS**

53035 The *pthread_condattr_init()* function shall fail if:

53036 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

53037 These functions shall not return an error code of [EINTR].

53038 **EXAMPLES**

53039 None.

53040 **APPLICATION USAGE**

53041 None.

53042 **RATIONALE**

53043 A *process-shared* attribute has been defined for condition variables for the same reason it has been
53044 defined for mutexes.

53045 If an implementation detects that the value specified by the *attr* argument to
53046 *pthread_condattr_destroy()* does not refer to an initialized condition variable attributes object, it is
53047 recommended that the function should fail and report an [EINVAL] error.

53048 See also *pthread_attr_destroy()* and *pthread_mutex_destroy()*.

53049 **FUTURE DIRECTIONS**

53050 None.

53051 **SEE ALSO**

53052 *pthread_attr_destroy()*, *pthread_cond_destroy()*, *pthread_condattr_getpshared()*, *pthread_create()*,
53053 *pthread_mutex_destroy()*

53054 XBD <pthread.h>

53055 **CHANGE HISTORY**

53056 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53057 **Issue 6**

53058 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are marked as part of the
53059 Threads option.

53060 **Issue 7**

53061 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are moved from the Threads
53062 option to the Base.

53063 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this
53064 condition results in undefined behavior.

53065 **NAME**

53066 pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection
53067 condition variable attribute

53068 **SYNOPSIS**

```
53069 #include <pthread.h>
53070 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
53071 clockid_t *restrict clock_id);
53072 int pthread_condattr_setclock(pthread_condattr_t *attr,
53073 clockid_t clock_id);
```

53074 **DESCRIPTION**

53075 The *pthread_condattr_getclock()* function shall obtain the value of the *clock* attribute from the
53076 attributes object referenced by *attr*.

53077 The *pthread_condattr_setclock()* function shall set the *clock* attribute in an initialized attributes
53078 object referenced by *attr*. If *pthread_condattr_setclock()* is called with a *clock_id* argument that
53079 refers to a CPU-time clock, the call shall fail.

53080 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of
53081 *pthread_cond_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

53082 The behavior is undefined if the value specified by the *attr* argument to
53083 *pthread_condattr_getclock()* or *pthread_condattr_setclock()* does not refer to an initialized condition
53084 variable attributes object.

53085 **RETURN VALUE**

53086 If successful, the *pthread_condattr_getclock()* function shall return zero and store the value of the
53087 clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error
53088 number shall be returned to indicate the error.

53089 If successful, the *pthread_condattr_setclock()* function shall return zero; otherwise, an error
53090 number shall be returned to indicate the error.

53091 **ERRORS**

53092 The *pthread_condattr_setclock()* function may fail if:

53093 [EINVAL] The value specified by *clock_id* does not refer to a known clock, or is a CPU-
53094 time clock.

53095 These functions shall not return an error code of [EINTR].

53096 **EXAMPLES**

53097 None.

53098 **APPLICATION USAGE**

53099 None.

53100 **RATIONALE**

53101 If an implementation detects that the value specified by the *attr* argument to
53102 *pthread_condattr_getclock()* or *pthread_condattr_setclock()* does not refer to an initialized condition
53103 variable attributes object, it is recommended that the function should fail and report an
53104 [EINVAL] error.

53105 **FUTURE DIRECTIONS**

53106 None.

53107 **SEE ALSO**

53108 *pthread_cond_destroy()*, *pthread_cond_timedwait()*, *pthread_condattr_destroy()*,
53109 *pthread_condattr_getpshared()*, *pthread_create()*, *pthread_mutex_destroy()*

53110 XBD <pthread.h>

53111 **CHANGE HISTORY**

53112 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53113 **Issue 7**

53114 The *pthread_condattr_getclock()* and *pthread_condattr_setclock()* functions are moved from the
53115 Clock Selection option to the Base.

53116 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this
53117 condition results in undefined behavior.

53118 **NAME**

53119 pthread_condattr_getpshared, pthread_condattr_setpshared ‡ get and set the process-shared
53120 condition variable attributes

53121 **SYNOPSIS**

```
53122 TSH #include <pthread.h>
53123
53123 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
53124 int *restrict pshared);
53125 int pthread_condattr_setpshared(pthread_condattr_t *attr,
53126 int pshared);
```

53127 **DESCRIPTION**

53128 The *pthread_condattr_getpshared()* function shall obtain the value of the *process-shared* attribute
53129 from the attributes object referenced by *attr*.

53130 The *pthread_condattr_setpshared()* function shall set the *process-shared* attribute in an initialized
53131 attributes object referenced by *attr*.

53132 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a condition
53133 variable to be operated upon by any thread that has access to the memory where the condition
53134 variable is allocated, even if the condition variable is allocated in memory that is shared by
53135 multiple processes. See [Section 2.9.9](#) (on page 523) for further requirements. The default value of
53136 the attribute is `PTHREAD_PROCESS_PRIVATE`.

53137 The behavior is undefined if the value specified by the *attr* argument to
53138 *pthread_condattr_getpshared()* or *pthread_condattr_setpshared()* does not refer to an initialized
53139 condition variable attributes object.

53140 **RETURN VALUE**

53141 If successful, the *pthread_condattr_setpshared()* function shall return zero; otherwise, an error
53142 number shall be returned to indicate the error.

53143 If successful, the *pthread_condattr_getpshared()* function shall return zero and store the value of
53144 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
53145 an error number shall be returned to indicate the error.

53146 **ERRORS**

53147 The *pthread_condattr_setpshared()* function may fail if:

53148 [EINVAL] The new value specified for the attribute is outside the range of legal values
53149 for that attribute.

53150 These functions shall not return an error code of [EINTR].

53151 **EXAMPLES**

53152 None.

53153 **APPLICATION USAGE**

53154 None.

53155 **RATIONALE**

53156 If an implementation detects that the value specified by the *attr* argument to
53157 *pthread_condattr_getpshared()* or *pthread_condattr_setpshared()* does not refer to an initialized
53158 condition variable attributes object, it is recommended that the function should fail and report
53159 an [EINVAL] error.

53160 **FUTURE DIRECTIONS**

53161 None.

53162 **SEE ALSO**53163 *pthread_create()*, *pthread_cond_destroy()*, *pthread_condattr_destroy()*, *pthread_mutex_destroy()*

53164 XBD <pthread.h>

53165 **CHANGE HISTORY**

53166 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53167 **Issue 6**53168 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are marked as part
53169 of the Threads and Thread Process-Shared Synchronization options.53170 The **restrict** keyword is added to the *pthread_condattr_getpshared()* prototype for alignment with
53171 the ISO/IEC 9899:1999 standard.53172 **Issue 7**53173 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are marked only as
53174 part of the Thread Process-Shared Synchronization option as the Threads option is now part of
53175 the Base.53176 The [EINVAL] error for an uninitialized condition variable attributes object is removed; this
53177 condition results in undefined behavior.53178 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0272 [972] and XSH/TC2-2008/0273
53179 [757] are applied.

53180 **NAME**

53181 pthread_condattr_init — initialize the condition variable attributes object

53182 **SYNOPSIS**

53183 #include <pthread.h>

53184 int pthread_condattr_init(pthread_condattr_t *attr);

53185 **DESCRIPTION**

53186 Refer to *pthread_condattr_destroy()*.

53187 **NAME**

53188 pthread_condattr_setclock — set the clock selection condition variable attribute

53189 **SYNOPSIS**

53190 #include <pthread.h>

53191 int pthread_condattr_setclock(pthread_condattr_t *attr,
53192 clockid_t clock_id);53193 **DESCRIPTION**53194 Refer to [pthread_condattr_getclock\(\)](#).

53195 **NAME**

53196 pthread_condattr_setpshared — set the process-shared condition variable attribute

53197 **SYNOPSIS**

```
53198 TSH #include <pthread.h>
53199 int pthread_condattr_setpshared(pthread_condattr_t *attr,
53200 int pshared);
```

53201 **DESCRIPTION**

53202 Refer to [pthread_condattr_getpshared\(\)](#).

53203 **NAME**

53204 pthread_create — thread creation

53205 **SYNOPSIS**

53206 #include <pthread.h>

```
53207 int pthread_create(pthread_t *restrict thread,
53208                  const pthread_attr_t *restrict attr,
53209                  void *(*start_routine)(void*), void *restrict arg);
```

53210 **DESCRIPTION**

53211 The *pthread_create()* function shall create a new thread, with attributes specified by *attr*, within a
 53212 process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are
 53213 modified later, the thread's attributes shall not be affected. Upon successful completion,
 53214 *pthread_create()* shall store the ID of the created thread in the location referenced by *thread*.

53215 The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine*
 53216 returns, the effect shall be as if there was an implicit call to *pthread_exit()* using the return value
 53217 of *start_routine* as the exit status. Note that the thread in which *main()* was originally invoked
 53218 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call to
 53219 *exit()* using the return value of *main()* as the exit status.

53220 The signal state of the new thread shall be initialized as follows:

53221 The signal mask shall be inherited from the creating thread.

53222 The set of signals pending for the new thread shall be empty.

53223 XSI The thread-local current locale and the alternate stack shall not be inherited.

53224 The floating-point environment shall be inherited from the creating thread.

53225 If *pthread_create()* fails, no new thread is created and the contents of the location referenced by
 53226 *thread* are undefined.

53227 TCT If `_POSIX_THREAD_CPUTIME` is defined, the new thread shall have a CPU-time clock
 53228 accessible, and the initial value of this clock shall be set to zero.

53229 The behavior is undefined if the value specified by the *attr* argument to *pthread_create()* does not
 53230 refer to an initialized thread attributes object.

53231 **RETURN VALUE**

53232 If successful, the *pthread_create()* function shall return zero; otherwise, an error number shall be
 53233 returned to indicate the error.

53234 **ERRORS**

53235 The *pthread_create()* function shall fail if:

53236 [EAGAIN] The system lacked the necessary resources to create another thread, or the
 53237 system-imposed limit on the total number of threads in a process
 53238 {PTHREAD_THREADS_MAX} would be exceeded.

53239 [EPERM] The caller does not have appropriate privileges to set the required scheduling
 53240 parameters or scheduling policy.

53241 The *pthread_create()* function shall not return an error code of [EINTR].

53242 **EXAMPLES**

53243 None.

53244 **APPLICATION USAGE**

53245 There is no requirement on the implementation that the ID of the created thread be available
53246 before the newly created thread starts executing. The calling thread can obtain the ID of the
53247 created thread through the *thread* argument of the *pthread_create()* function, and the newly
53248 created thread can obtain its ID by a call to *pthread_self()*.

53249 **RATIONALE**

53250 A suggested alternative to *pthread_create()* would be to define two separate operations: create
53251 and start. Some applications would find such behavior more natural. Ada, in particular,
53252 separates the "creation" of a task from its "activation".

53253 Splitting the operation was rejected by the standard developers for many reasons:

53254 The number of calls required to start a thread would increase from one to two and thus
53255 place an additional burden on applications that do not require the additional
53256 synchronization. The second call, however, could be avoided by the additional
53257 complication of a start-up state attribute.

53258 An extra state would be introduced: "created but not started". This would require the
53259 standard to specify the behavior of the thread operations when the target has not yet
53260 started executing.

53261 For those applications that require such behavior, it is possible to simulate the two separate
53262 steps with the facilities that are currently provided. The *start_routine()* can synchronize by
53263 waiting on a condition variable that is signaled by the start operation.

53264 An Ada implementor can choose to create the thread at either of two points in the Ada program:
53265 when the task object is created, or when the task is activated (generally at a "begin"). If the first
53266 approach is adopted, the *start_routine()* needs to wait on a condition variable to receive the order
53267 to begin "activation". The second approach requires no such condition variable or extra
53268 synchronization. In either approach, a separate Ada task control block would need to be created
53269 when the task object is created to hold rendezvous queues, and so on.

53270 An extension of the preceding model would be to allow the state of the thread to be modified
53271 between the create and start. This would allow the thread attributes object to be eliminated. This
53272 has been rejected because:

53273 All state in the thread attributes object has to be able to be set for the thread. This would
53274 require the definition of functions to modify thread attributes. There would be no
53275 reduction in the number of function calls required to set up the thread. In fact, for an
53276 application that creates all threads using identical attributes, the number of function calls
53277 required to set up the threads would be dramatically increased. Use of a thread attributes
53278 object permits the application to make one set of attribute setting function calls.
53279 Otherwise, the set of attribute setting function calls needs to be made for each thread
53280 creation.

53281 Depending on the implementation architecture, functions to set thread state would require
53282 kernel calls, or for other implementation reasons would not be able to be implemented as
53283 macros, thereby increasing the cost of thread creation.

53284 The ability for applications to segregate threads by class would be lost.

53285 Another suggested alternative uses a model similar to that for process creation, such as "thread
53286 fork". The fork semantics would provide more flexibility and the "create" function can be
53287 implemented simply by doing a thread fork followed immediately by a call to the desired "start

- 53288 routine” for the thread. This alternative has these problems:
- 53289 For many implementations, the entire stack of the calling thread would need to be
53290 duplicated, since in many architectures there is no way to determine the size of the calling
53291 frame.
- 53292 Efficiency is reduced since at least some part of the stack has to be copied, even though in
53293 most cases the thread never needs the copied context, since it merely calls the desired start
53294 routine.
- 53295 If an implementation detects that the value specified by the *attr* argument to *pthread_create()*
53296 does not refer to an initialized thread attributes object, it is recommended that the function
53297 should fail and report an [EINVAL] error.
- 53298 **FUTURE DIRECTIONS**
- 53299 None.
- 53300 **SEE ALSO**
- 53301 *fork()*, *pthread_exit()*, *pthread_join()*
- 53302 XBD Section 4.12 (on page 111), <[pthread.h](#)>
- 53303 **CHANGE HISTORY**
- 53304 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
- 53305 **Issue 6**
- 53306 The *pthread_create()* function is marked as part of the Threads option.
- 53307 The following new requirements on POSIX implementations derive from alignment with the
53308 Single UNIX Specification:
- 53309 The [EPERM] mandatory error condition is added.
- 53310 The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.
- 53311 The **restrict** keyword is added to the *pthread_create()* prototype for alignment with the
53312 ISO/IEC 9899:1999 standard.
- 53313 The DESCRIPTION is updated to make it explicit that the floating-point environment is
53314 inherited from the creating thread.
- 53315 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the
53316 alternate stack is not inherited.
- 53317 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/93 is applied, updating the ERRORS
53318 section to remove the mandatory [EINVAL] error (“The value specified by *attr* is invalid”), and
53319 adding the optional [EINVAL] error (“The attributes specified by *attr* are invalid”).
- 53320 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/94 is applied, adding the APPLICATION
53321 USAGE section.
- 53322 **Issue 7**
- 53323 The *pthread_create()* function is moved from the Threads option to the Base.
- 53324 The [EINVAL] error for an uninitialized thread attributes object is removed; this condition
53325 results in undefined behavior.
- 53326 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0458 [302] is applied.
- 53327 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0274 [849] is applied.

53328 **NAME**

53329 pthread_detach — detach a thread

53330 **SYNOPSIS**

53331 #include <pthread.h>

53332 int pthread_detach(pthread_t thread);

53333 **DESCRIPTION**

53334 The *pthread_detach()* function shall indicate to the implementation that storage for the thread
53335 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,
53336 *pthread_detach()* shall not cause it to terminate.

53337 The behavior is undefined if the value specified by the *thread* argument to *pthread_detach()* does
53338 not refer to a joinable thread.

53339 **RETURN VALUE**

53340 If the call succeeds, *pthread_detach()* shall return 0; otherwise, an error number shall be returned
53341 to indicate the error.

53342 **ERRORS**

53343 The *pthread_detach()* function shall not return an error code of [EINTR].

53344 **EXAMPLES**

53345 None.

53346 **APPLICATION USAGE**

53347 None.

53348 **RATIONALE**

53349 The *pthread_join()* or *pthread_detach()* functions should eventually be called for every thread that
53350 is created so that storage associated with the thread may be reclaimed.

53351 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation
53352 attribute is sufficient, since a thread need never be dynamically detached. However, need arises
53353 in at least two cases:

53354 1. In a cancellation handler for a *pthread_join()* it is nearly essential to have a
53355 *pthread_detach()* function in order to detach the thread on which *pthread_join()* was
53356 waiting. Without it, it would be necessary to have the handler do another *pthread_join()* to
53357 attempt to detach the thread, which would both delay the cancellation processing for an
53358 unbounded period and introduce a new call to *pthread_join()*, which might itself need a
53359 cancellation handler. A dynamic detach is nearly essential in this case.

53360 2. In order to detach the “initial thread” (as may be desirable in processes that set up server
53361 threads).

53362 If an implementation detects that the value specified by the *thread* argument to *pthread_detach()*
53363 does not refer to a joinable thread, it is recommended that the function should fail and report an
53364 [EINVAL] error.

53365 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
53366 that the function should fail and report an [ESRCH] error.

53367 **FUTURE DIRECTIONS**

53368 None.

53369 **SEE ALSO**53370 [pthread_join\(\)](#)53371 XBD <[pthread.h](#)>53372 **CHANGE HISTORY**

53373 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53374 **Issue 6**53375 The *pthread_detach()* function is marked as part of the Threads option.53376 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/95 is applied, updating the ERRORS
53377 section so that the [EINVAL] and [ESRCH] error cases become optional.53378 **Issue 7**53379 The *pthread_detach()* function is moved from the Threads option to the Base.

53380 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

53381 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined
53382 behavior.

53383 **NAME**

53384 pthread_equal — compare thread IDs

53385 **SYNOPSIS**

53386 #include <pthread.h>

53387 int pthread_equal(pthread_t t1, pthread_t t2);

53388 **DESCRIPTION**53389 This function shall compare the thread IDs *t1* and *t2*.53390 **RETURN VALUE**53391 The *pthread_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero
53392 shall be returned.53393 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.53394 **ERRORS**

53395 No errors are defined.

53396 The *pthread_equal()* function shall not return an error code of [EINTR].53397 **EXAMPLES**

53398 None.

53399 **APPLICATION USAGE**

53400 None.

53401 **RATIONALE**53402 Implementations may choose to define a thread ID as a structure. This allows additional
53403 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence
53404 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).
53405 Since the C language does not support comparison on structure types, the *pthread_equal()*
53406 function is provided to compare thread IDs.53407 **FUTURE DIRECTIONS**

53408 None.

53409 **SEE ALSO**53410 *pthread_create()*, *pthread_self()*

53411 XBD <pthread.h>

53412 **CHANGE HISTORY**

53413 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53414 **Issue 6**53415 The *pthread_equal()* function is marked as part of the Threads option.53416 **Issue 7**53417 The *pthread_equal()* function is moved from the Threads option to the Base.

53418 **NAME**

53419 pthread_exit — thread termination

53420 **SYNOPSIS**

53421 #include <pthread.h>

53422 void pthread_exit(void *value_ptr);

53423 **DESCRIPTION**

53424 The *pthread_exit()* function shall terminate the calling thread and make the value *value_ptr*
53425 available to any successful join with the terminating thread. Any cancellation cleanup handlers
53426 that have been pushed and not yet popped shall be popped in the reverse order that they were
53427 pushed and then executed. After all cancellation cleanup handlers have been executed, if the
53428 thread has any thread-specific data, appropriate destructor functions shall be called in an
53429 unspecified order. Thread termination does not release any application visible process resources,
53430 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level
53431 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

53432 An implicit call to *pthread_exit()* is made when a thread other than the thread in which *main()*
53433 was first invoked returns from the start routine that was used to create it. The function's return
53434 value shall serve as the thread's exit status.

53435 The behavior of *pthread_exit()* is undefined if called from a cancellation cleanup handler or
53436 destructor function that was invoked as a result of either an implicit or explicit call to
53437 *pthread_exit()*.

53438 After a thread has terminated, the result of access to local (auto) variables of the thread is
53439 undefined. Thus, references to local variables of the exiting thread should not be used for the
53440 *pthread_exit()* *value_ptr* parameter value.

53441 The process shall exit with an exit status of 0 after the last thread has been terminated. The
53442 behavior shall be as if the implementation called *exit()* with a zero argument at thread
53443 termination time.

53444 **RETURN VALUE**53445 The *pthread_exit()* function cannot return to its caller.53446 **ERRORS**

53447 No errors are defined.

53448 **EXAMPLES**

53449 None.

53450 **APPLICATION USAGE**

53451 None.

53452 **RATIONALE**

53453 The normal mechanism by which a thread terminates is to return from the routine that was
53454 specified in the *pthread_create()* call that started it. The *pthread_exit()* function provides the
53455 capability for a thread to terminate without requiring a return from the start routine of that
53456 thread, thereby providing a function analogous to *exit()*.

53457 Regardless of the method of thread termination, any cancellation cleanup handlers that have
53458 been pushed and not yet popped are executed, and the destructors for any existing thread-
53459 specific data are executed. This volume of POSIX.1-2017 requires that cancellation cleanup
53460 handlers be popped and called in order. After all cancellation cleanup handlers have been
53461 executed, thread-specific data destructors are called, in an unspecified order, for each item of
53462 thread-specific data that exists in the thread. This ordering is necessary because cancellation
53463 cleanup handlers may rely on thread-specific data.

53464 As the meaning of the status is determined by the application (except when the thread has been
53465 canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an
53466 illegal status value is, which is why no address error checking is done.

53467 **FUTURE DIRECTIONS**

53468 None.

53469 **SEE ALSO**

53470 *exit()*, *pthread_create()*, *pthread_join()*

53471 XBD <pthread.h>

53472 **CHANGE HISTORY**

53473 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53474 **Issue 6**

53475 The *pthread_exit()* function is marked as part of the Threads option.

53476 **Issue 7**

53477 The *pthread_exit()* function is moved from the Threads option to the Base.

53478 **NAME**

53479 pthread_getconcurrency, pthread_setconcurrency — get and set the level of concurrency

53480 **SYNOPSIS**

```
53481 OB XSI #include <pthread.h>
53482 int pthread_getconcurrency(void);
53483 int pthread_setconcurrency(int new_level);
```

53484 **DESCRIPTION**

53485 Unbound threads in a process may or may not be required to be simultaneously active. By
 53486 default, the threads implementation ensures that a sufficient number of threads are active so that
 53487 the process can continue to make progress. While this conserves system resources, it may not
 53488 produce the most effective level of concurrency.

53489 The *pthread_setconcurrency()* function allows an application to inform the threads
 53490 implementation of its desired concurrency level, *new_level*. The actual level of concurrency
 53491 provided by the implementation as a result of this function call is unspecified.

53492 If *new_level* is zero, it causes the implementation to maintain the concurrency level at its
 53493 discretion as if *pthread_setconcurrency()* had never been called.

53494 The *pthread_getconcurrency()* function shall return the value set by a previous call to the
 53495 *pthread_setconcurrency()* function. If the *pthread_setconcurrency()* function was not previously
 53496 called, this function shall return zero to indicate that the implementation is maintaining the
 53497 concurrency level.

53498 A call to *pthread_setconcurrency()* shall inform the implementation of its desired concurrency
 53499 level. The implementation shall use this as a hint, not a requirement.

53500 If an implementation does not support multiplexing of user threads on top of several kernel-
 53501 scheduled entities, the *pthread_setconcurrency()* and *pthread_getconcurrency()* functions are
 53502 provided for source code compatibility but they shall have no effect when called. To maintain
 53503 the function semantics, the *new_level* parameter is saved when *pthread_setconcurrency()* is called
 53504 so that a subsequent call to *pthread_getconcurrency()* shall return the same value.

53505 **RETURN VALUE**

53506 If successful, the *pthread_setconcurrency()* function shall return zero; otherwise, an error number
 53507 shall be returned to indicate the error.

53508 The *pthread_getconcurrency()* function shall always return the concurrency level set by a previous
 53509 call to *pthread_setconcurrency()*. If the *pthread_setconcurrency()* function has never been called,
 53510 *pthread_getconcurrency()* shall return zero.

53511 **ERRORS**

53512 The *pthread_setconcurrency()* function shall fail if:

53513 [EINVAL] The value specified by *new_level* is negative.

53514 [EAGAIN] The value specified by *new_level* would cause a system resource to be
 53515 exceeded.

53516 The *pthread_setconcurrency()* function shall not return an error code of [EINTR].

53517 **EXAMPLES**
53518 None.

53519 **APPLICATION USAGE**

53520 Application developers should note that an implementation can always ignore any calls to
53521 *pthread_setconcurrency()* and return a constant for *pthread_getconcurrency()*. For this reason, it is
53522 not recommended that portable applications use this function.

53523 **RATIONALE**

53524 None.

53525 **FUTURE DIRECTIONS**

53526 These functions may be removed in a future version.

53527 **SEE ALSO**

53528 XBD <[pthread.h](#)>

53529 **CHANGE HISTORY**

53530 First released in Issue 5.

53531 **Issue 7**

53532 SD5-XSH-ERN-184 is applied.

53533 The *pthread_getconcurrency()* and *pthread_setconcurrency()* functions are marked obsolescent.

53534 **NAME**

53535 pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME**
53536 **THREADS**)

53537 **SYNOPSIS**

```
53538 TCT #include <pthread.h>
53539 #include <time.h>
53540 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

53541 **DESCRIPTION**

53542 The *pthread_getcpuclockid()* function shall return in *clock_id* the clock ID of the CPU-time clock of
53543 the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

53544 **RETURN VALUE**

53545 Upon successful completion, *pthread_getcpuclockid()* shall return zero; otherwise, an error
53546 number shall be returned to indicate the error.

53547 **ERRORS**

53548 No errors are defined.

53549 **EXAMPLES**

53550 None.

53551 **APPLICATION USAGE**

53552 The *pthread_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not
53553 be provided on all implementations.

53554 **RATIONALE**

53555 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
53556 that the function should fail and report an [ESRCH] error.

53557 **FUTURE DIRECTIONS**

53558 None.

53559 **SEE ALSO**

53560 *clock_getcpuclockid()*, *clock_getres()*, *timer_create()*

53561 XBD [<pthread.h>](#), [<time.h>](#)

53562 **CHANGE HISTORY**

53563 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

53564 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

53565 **Issue 7**

53566 The *pthread_getcpuclockid()* function is marked only as part of the Thread CPU-Time Clocks
53567 option as the Threads option is now part of the Base.

53568 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

53569 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0275 [757] is applied.

53570 NAME

53571 pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters
53572 access (REALTIME THREADS)

53573 SYNOPSIS

```
53574 TPS #include <pthread.h>
53575
53575 int pthread_getschedparam(pthread_t thread, int *restrict policy,
53576 struct sched_param *restrict param);
53577
53577 int pthread_setschedparam(pthread_t thread, int policy,
53578 const struct sched_param *param);
```

53579 DESCRIPTION

53580 The *pthread_getschedparam()* and *pthread_setschedparam()* functions shall, respectively, get and set
53581 the scheduling policy and parameters of individual threads within a multi-threaded process to
53582 be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the
53583 **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected
53584 scheduling parameters are implementation-defined.

53585 The *pthread_getschedparam()* function shall retrieve the scheduling policy and scheduling
53586 parameters for the thread whose thread ID is given by *thread* and shall store those values in
53587 *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam()* shall be
53588 the value specified by the most recent *pthread_setschedparam()*, *pthread_setschedprio()*, or
53589 *pthread_create()* call affecting the target thread. It shall not reflect any temporary adjustments to
53590 its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam()*
53591 function shall set the scheduling policy and associated scheduling parameters for the thread
53592 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*
53593 and *param*, respectively.

53594 The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The
53595 scheduling parameters for the SCHED_OTHER policy are implementation-defined. The
53596 SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

53597 TSP If _POSIX_THREAD_SPORADIC_SERVER is defined, then the *policy* argument may have the
53598 value SCHED_SPORADIC, with the exception for the *pthread_setschedparam()* function that if the
53599 scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-
53600 defined whether the function is supported; in other words, the implementation need not allow
53601 the application to dynamically change the scheduling policy to SCHED_SPORADIC. The
53602 sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*,
53603 *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified
53604 *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the
53605 function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall
53606 be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function
53607 shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are
53608 stored as provided by this function or are rounded to align with the resolution of the clock being
53609 used.

53610 If the *pthread_setschedparam()* function fails, the scheduling parameters shall not be changed for
53611 the target thread.

53612 RETURN VALUE

53613 If successful, the *pthread_getschedparam()* and *pthread_setschedparam()* functions shall return zero;
53614 otherwise, an error number shall be returned to indicate the error.

53615 **ERRORS**53616 The *pthread_setschedparam()* function shall fail if:53617 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an
53618 unsupported value.53619 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to
53620 SCHED_SPORADIC, and the implementation does not support this change.53621 The *pthread_setschedparam()* function may fail if:53622 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated
53623 with the scheduling policy *policy* is invalid.53624 [EPERM] The caller does not have appropriate privileges to set either the scheduling
53625 parameters or the scheduling policy of the specified thread.53626 [EPERM] The implementation does not allow the application to modify one of the
53627 parameters to the value specified.

53628 These functions shall not return an error code of [EINTR].

53629 **EXAMPLES**

53630 None.

53631 **APPLICATION USAGE**

53632 None.

53633 **RATIONALE**53634 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
53635 that the function should fail and report an [ESRCH] error.53636 **FUTURE DIRECTIONS**

53637 None.

53638 **SEE ALSO**53639 *pthread_setschedprio()*, *sched_getparam()*, *sched_getscheduler()*

53640 XBD <pthread.h>, <sched.h>

53641 **CHANGE HISTORY**

53642 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53643 **Issue 6**53644 The *pthread_getschedparam()* and *pthread_setschedparam()* functions are marked as part of the
53645 Threads and Thread Execution Scheduling options.53646 The [ENOSYS] error condition has been removed as stubs need not be provided if an
53647 implementation does not support the Thread Execution Scheduling option.53648 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
53649 *pthread_setschedparam()* function so that its second argument is of type **int**.

53650 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

53651 The **restrict** keyword is added to the *pthread_getschedparam()* prototype for alignment with the
53652 ISO/IEC 9899:1999 standard.

53653 The Open Group Corrigendum U047/1 is applied.

53654 IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a
53655 call to the *pthread_setschedprio()* function.

53656 **Issue 7**

53657 The *pthread_getschedparam()* and *pthread_setschedparam()* functions are marked only as part of the
53658 Thread Execution Scheduling option as the Threads option is now part of the Base.

53659 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
53660 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

53661 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

53662 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0459 [314] is applied.

53663 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0276 [757] is applied.

53664 **NAME**

53665 pthread_getspecific, pthread_setspecific — thread-specific data management

53666 **SYNOPSIS**

53667 #include <pthread.h>

53668 void *pthread_getspecific(pthread_key_t key);

53669 int pthread_setspecific(pthread_key_t key, const void *value);

53670 **DESCRIPTION**53671 The *pthread_getspecific()* function shall return the value currently bound to the specified *key* on
53672 behalf of the calling thread.53673 The *pthread_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a
53674 previous call to *pthread_key_create()*. Different threads may bind different values to the same
53675 key. These values are typically pointers to blocks of dynamically allocated memory that have
53676 been reserved for use by the calling thread.53677 The effect of calling *pthread_getspecific()* or *pthread_setspecific()* with a *key* value not obtained
53678 from *pthread_key_create()* or after *key* has been deleted with *pthread_key_delete()* is undefined.53679 Both *pthread_getspecific()* and *pthread_setspecific()* may be called from a thread-specific data
53680 destructor function. A call to *pthread_getspecific()* for the thread-specific data key being
53681 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)
53682 by a call to *pthread_setspecific()*. Calling *pthread_setspecific()* from a thread-specific data
53683 destructor routine may result either in lost storage (after at least
53684 PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

53685 Both functions may be implemented as macros.

53686 **RETURN VALUE**53687 The *pthread_getspecific()* function shall return the thread-specific data value associated with the
53688 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be
53689 returned.53690 If successful, the *pthread_setspecific()* function shall return zero; otherwise, an error number shall
53691 be returned to indicate the error.53692 **ERRORS**53693 No errors are returned from *pthread_getspecific()*.53694 The *pthread_setspecific()* function shall fail if:

53695 [ENOMEM] Insufficient memory exists to associate the non-NULL value with the key.

53696 The *pthread_setspecific()* function shall not return an error code of [EINTR].53697 **EXAMPLES**

53698 None.

53699 **APPLICATION USAGE**

53700 None.

53701 **RATIONALE**53702 Performance and ease-of-use of *pthread_getspecific()* are critical for functions that rely on
53703 maintaining state in thread-specific data. Since no errors are required to be detected by it, and
53704 since the only error that could be detected is the use of an invalid key, the function to
53705 *pthread_getspecific()* has been designed to favor speed and simplicity over error reporting.53706 If an implementation detects that the value specified by the *key* argument to *pthread_setspecific()*
53707 does not refer to a key value obtained from *pthread_key_create()* or refers to a key that has been

53708 deleted with *pthread_key_delete()*, it is recommended that the function should fail and report an
53709 [EINVAL] error.

53710 FUTURE DIRECTIONS

53711 None.

53712 SEE ALSO

53713 *pthread_key_create()*

53714 XBD <pthread.h>

53715 CHANGE HISTORY

53716 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53717 Issue 6

53718 The *pthread_getspecific()* and *pthread_setspecific()* functions are marked as part of the Threads
53719 option.

53720 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

53721 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/96 is applied, updating the ERRORS
53722 section so that the [ENOMEM] error case is changed from ``to associate the value with the key''
53723 to ``to associate the non-NULL value with the key''.

53724 Issue 7

53725 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the ERRORS section.

53726 The *pthread_getspecific()* and *pthread_setspecific()* functions are moved from the Threads option to
53727 the Base.

53728 The [EINVAL] error for a key value not obtained from *pthread_key_create()* or a key deleted with
53729 *pthread_key_delete()* is removed; this condition results in undefined behavior.

53730 **NAME**

53731 pthread_join — wait for thread termination

53732 **SYNOPSIS**

53733 #include <pthread.h>

53734 int pthread_join(pthread_t *thread*, void ***value_ptr*);53735 **DESCRIPTION**

53736 The *pthread_join()* function shall suspend execution of the calling thread until the target *thread*
 53737 terminates, unless the target *thread* has already terminated. On return from a successful
 53738 *pthread_join()* call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit()* by
 53739 the terminating thread shall be made available in the location referenced by *value_ptr*. When a
 53740 *pthread_join()* returns successfully, the target thread has been terminated. The results of multiple
 53741 simultaneous calls to *pthread_join()* specifying the same target thread are undefined. If the
 53742 thread calling *pthread_join()* is canceled, then the target thread shall not be detached.

53743 It is unspecified whether a thread that has exited but remains unjoined counts against
 53744 {PTHREAD_THREADS_MAX}.

53745 The behavior is undefined if the value specified by the *thread* argument to *pthread_join()* does not
 53746 refer to a joinable thread.

53747 The behavior is undefined if the value specified by the *thread* argument to *pthread_join()* refers to
 53748 the calling thread.

53749 **RETURN VALUE**

53750 If successful, the *pthread_join()* function shall return zero; otherwise, an error number shall be
 53751 returned to indicate the error.

53752 **ERRORS**53753 The *pthread_join()* function may fail if:

53754 [EDEADLK] A deadlock was detected.

53755 The *pthread_join()* function shall not return an error code of [EINTR].53756 **EXAMPLES**

53757 An example of thread creation and deletion follows:

```

53758 typedef struct {
53759     int *ar;
53760     long n;
53761 } subarray;

53762 void *
53763 incer(void *arg)
53764 {
53765     long i;

53766     for (i = 0; i < ((subarray *)arg)->n; i++)
53767         ((subarray *)arg)->ar[i]++;
53768 }

53769 int main(void)
53770 {
53771     int         ar[1000000];
53772     pthread_t   th1, th2;
53773     subarray    sb1, sb2;

```



```

53774         sb1.ar = &ar[0];
53775         sb1.n  = 500000;
53776         (void) pthread_create(&th1, NULL, incer, &sb1);

53777         sb2.ar = &ar[500000];
53778         sb2.n  = 500000;
53779         (void) pthread_create(&th2, NULL, incer, &sb2);

53780         (void) pthread_join(th1, NULL);
53781         (void) pthread_join(th2, NULL);
53782         return 0;
53783     }

```

53784 APPLICATION USAGE

53785 None.

53786 RATIONALE

53787 The *pthread_join()* function is a convenience that has proven useful in multi-threaded
53788 applications. It is true that a programmer could simulate this function if it were not provided by
53789 passing extra state as part of the argument to the *start_routine()*. The terminating thread would
53790 set a flag to indicate termination and broadcast a condition that is part of that state; a joining
53791 thread would wait on that condition variable. While such a technique would allow a thread to
53792 wait on more complex conditions (for example, waiting for multiple threads to terminate),
53793 waiting on individual thread termination is considered widely useful. Also, including the
53794 *pthread_join()* function in no way precludes a programmer from coding such complex waits.
53795 Thus, while not a primitive, including *pthread_join()* in this volume of POSIX.1-2017 was
53796 considered valuable.

53797 The *pthread_join()* function provides a simple mechanism allowing an application to wait for a
53798 thread to terminate. After the thread terminates, the application may then choose to clean up
53799 resources that were used by the thread. For instance, after *pthread_join()* returns, any
53800 application-provided stack storage could be reclaimed.

53801 The *pthread_join()* or *pthread_detach()* function should eventually be called for every thread that
53802 is created with the *detachstate* attribute set to *PTHREAD_CREATE_JOINABLE* so that storage
53803 associated with the thread may be reclaimed.

53804 The interaction between *pthread_join()* and cancellation is well-defined for the following reasons:

53805 The *pthread_join()* function, like all other non-async-cancel-safe functions, can only be
53806 called with deferred cancelability type.

53807 Cancellation cannot occur in the disabled cancelability state.

53808 Thus, only the default cancelability state need be considered. As specified, either the
53809 *pthread_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the
53810 application, since either a cancellation handler is run or *pthread_join()* returns. There are no race
53811 conditions since *pthread_join()* was called in the deferred cancelability state.

53812 If an implementation detects that the value specified by the *thread* argument to *pthread_join()*
53813 does not refer to a joinable thread, it is recommended that the function should fail and report an
53814 [EINVAL] error.

53815 If an implementation detects that the value specified by the *thread* argument to *pthread_join()*
53816 refers to the calling thread, it is recommended that the function should fail and report an
53817 [EDEADLK] error.

53818 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended

53819 that the function should fail and report an [ESRCH] error.

53820 **FUTURE DIRECTIONS**

53821 None.

53822 **SEE ALSO**

53823 *pthread_create()*, *wait()*

53824 XBD Section 4.12 (on page 111), <pthread.h>

53825 **CHANGE HISTORY**

53826 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53827 **Issue 6**

53828 The *pthread_join()* function is marked as part of the Threads option.

53829 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/97 is applied, updating the ERRORS
53830 section so that the [EINVAL] error is made optional and the words “the implementation has
53831 detected” are removed from it.

53832 **Issue 7**

53833 The *pthread_join()* function is moved from the Threads option to the Base.

53834 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

53835 The [EINVAL] error for a non-joinable thread is removed; this condition results in undefined
53836 behavior.

53837 The [EDEADLK] error for the calling thread is removed; this condition results in undefined
53838 behavior.

53839 **NAME**

53840 pthread_key_create — thread-specific data key creation

53841 **SYNOPSIS**

53842 #include <pthread.h>

53843 int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));

53844 **DESCRIPTION**

53845 The *pthread_key_create()* function shall create a thread-specific data key visible to all threads in
 53846 the process. Key values provided by *pthread_key_create()* are opaque objects used to locate
 53847 thread-specific data. Although the same key value may be used by different threads, the values
 53848 bound to the key by *pthread_setspecific()* are maintained on a per-thread basis and persist for the
 53849 life of the calling thread.

53850 Upon key creation, the value NULL shall be associated with the new key in all active threads.
 53851 Upon thread creation, the value NULL shall be associated with all defined keys in the new
 53852 thread.

53853 An optional destructor function may be associated with each key value. At thread exit, if a key
 53854 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with
 53855 that key, the value of the key is set to NULL, and then the function pointed to is called with the
 53856 previously associated value as its sole argument. The order of destructor calls is unspecified if
 53857 more than one destructor exists for a thread when it exits.

53858 If, after all the destructors have been called for all non-NULL values with associated destructors,
 53859 there are still some non-NULL values with associated destructors, then the process is repeated.
 53860 If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for
 53861 outstanding non-NULL values, there are still some non-NULL values with associated
 53862 destructors, implementations may stop calling destructors, or they may continue calling
 53863 destructors until no non-NULL values with associated destructors exist, even though this might
 53864 result in an infinite loop.

53865 **RETURN VALUE**

53866 If successful, the *pthread_key_create()* function shall store the newly created key value at **key* and
 53867 shall return zero. Otherwise, an error number shall be returned to indicate the error.

53868 **ERRORS**53869 The *pthread_key_create()* function shall fail if:

53870 [EAGAIN] The system lacked the necessary resources to create another thread-specific
 53871 data key, or the system-imposed limit on the total number of keys per process
 53872 {PTHREAD_KEYS_MAX} has been exceeded.

53873 [ENOMEM] Insufficient memory exists to create the key.

53874 The *pthread_key_create()* function shall not return an error code of [EINTR].53875 **EXAMPLES**

53876 The following example demonstrates a function that initializes a thread-specific data key when it
 53877 is first called, and associates a thread-specific object with each calling thread, initializing this
 53878 object when necessary.

```
53879 static pthread_key_t key;
53880 static pthread_once_t key_once = PTHREAD_ONCE_INIT;

53881 static void
53882 make_key()
53883 {
```

```

53884     (void) pthread_key_create(&key, NULL);
53885 }
53886 func()
53887 {
53888     void *ptr;
53889     (void) pthread_once(&key_once, make_key);
53890     if ((ptr = pthread_getspecific(key)) == NULL) {
53891         ptr = malloc(OBJECT_SIZE);
53892         ...
53893         (void) pthread_setspecific(key, ptr);
53894     }
53895     ...
53896 }

```

53897 Note that the key has to be initialized before *pthread_getspecific()* or *pthread_setspecific()* can be
53898 used. The *pthread_key_create()* call could either be explicitly made in a module initialization
53899 routine, or it can be done implicitly by the first call to a module as in this example. Any attempt
53900 to use the key before it is initialized is a programming error, making the code below incorrect.

```

53901 static pthread_key_t key;
53902 func()
53903 {
53904     void *ptr;
53905     /* KEY NOT INITIALIZED!!! THIS WILL NOT WORK!!! */
53906     if ((ptr = pthread_getspecific(key)) == NULL &&
53907         pthread_setspecific(key, NULL) != 0) {
53908         pthread_key_create(&key, NULL);
53909         ...
53910     }
53911 }

```

53912 APPLICATION USAGE

53913 None.

53914 RATIONALE

53915 Destructor Functions

53916 Normally, the value bound to a key on behalf of a particular thread is a pointer to storage
53917 allocated dynamically on behalf of the calling thread. The destructor functions specified with
53918 *pthread_key_create()* are intended to be used to free this storage when the thread exits. Thread
53919 cancellation cleanup handlers cannot be used for this purpose because thread-specific data may
53920 persist outside the lexical scope in which the cancellation cleanup handlers operate.

53921 If the value associated with a key needs to be updated during the lifetime of the thread, it may
53922 be necessary to release the storage associated with the old value before the new value is bound.
53923 Although the *pthread_setspecific()* function could do this automatically, this feature is not needed
53924 often enough to justify the added complexity. Instead, the programmer is responsible for freeing
53925 the stale storage:

```

53926 pthread_getspecific(key, &old);
53927 new = allocate();
53928 destructor(old);

```

53929 pthread_setspecific(key, new);

53930 **Note:** The above example could leak storage if run with asynchronous cancellation enabled. No such
53931 problems occur in the default cancellation state if no cancellation points occur between the get
53932 and set.

53933 There is no notion of a destructor-safe function. If an application does not call *pthread_exit()*
53934 from a signal handler, or if it blocks any signal whose handler may call *pthread_exit()* while
53935 calling async-unsafe functions, all functions may be safely called from destructors.

53936 Non-Idempotent Data Key Creation

53937 There were requests to make *pthread_key_create()* idempotent with respect to a given *key* address
53938 parameter. This would allow applications to call *pthread_key_create()* multiple times for a given
53939 *key* address and be guaranteed that only one key would be created. Doing so would require the
53940 key value to be previously initialized (possibly at compile time) to a known null value and
53941 would require that implicit mutual-exclusion be performed based on the address and contents of
53942 the *key* parameter in order to guarantee that exactly one key would be created.

53943 Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create()*.
53944 On many implementations, implicit mutual-exclusion would also have to be performed by
53945 *pthread_getspecific()* and *pthread_setspecific()* in order to guard against using incompletely stored
53946 or not-yet-visible key values. This could significantly increase the cost of important operations,
53947 particularly *pthread_getspecific()*.

53948 Thus, this proposal was rejected. The *pthread_key_create()* function performs no implicit
53949 synchronization. It is the responsibility of the programmer to ensure that it is called exactly once
53950 per key before use of the key. Several straightforward mechanisms can already be used to
53951 accomplish this, including calling explicit module initialization functions, using mutexes, and
53952 using *pthread_once()*. This places no significant burden on the programmer, introduces no
53953 possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows
53954 commonly used thread-specific data operations to be more efficient.

53955 FUTURE DIRECTIONS

53956 None.

53957 SEE ALSO

53958 [*pthread_getspecific\(\)*](#), [*pthread_key_delete\(\)*](#)

53959 XBD <[pthread.h](#)>

53960 CHANGE HISTORY

53961 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53962 Issue 6

53963 The *pthread_key_create()* function is marked as part of the Threads option.

53964 IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

53965 Issue 7

53966 The *pthread_key_create()* function is moved from the Threads option to the Base.

53967 **NAME**

53968 pthread_key_delete — thread-specific data key deletion

53969 **SYNOPSIS**

53970 #include <pthread.h>

53971 int pthread_key_delete(pthread_key_t key);

53972 **DESCRIPTION**

53973 The *pthread_key_delete()* function shall delete a thread-specific data key previously returned by
 53974 *pthread_key_create()*. The thread-specific data values associated with *key* need not be NULL at
 53975 the time *pthread_key_delete()* is called. It is the responsibility of the application to free any
 53976 application storage or perform any cleanup actions for data structures related to the deleted key
 53977 or associated thread-specific data in any threads; this cleanup can be done either before or after
 53978 *pthread_key_delete()* is called. Any attempt to use *key* following the call to *pthread_key_delete()*
 53979 results in undefined behavior.

53980 The *pthread_key_delete()* function shall be callable from within destructor functions. No
 53981 destructor functions shall be invoked by *pthread_key_delete()*. Any destructor function that may
 53982 have been associated with *key* shall no longer be called upon thread exit.

53983 **RETURN VALUE**

53984 If successful, the *pthread_key_delete()* function shall return zero; otherwise, an error number shall
 53985 be returned to indicate the error.

53986 **ERRORS**53987 The *pthread_key_delete()* function shall not return an error code of [EINTR].53988 **EXAMPLES**

53989 None.

53990 **APPLICATION USAGE**

53991 None.

53992 **RATIONALE**

53993 A thread-specific data key deletion function has been included in order to allow the resources
 53994 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys
 53995 can arise, among other scenarios, when a dynamically loaded module that allocated a key is
 53996 unloaded.

53997 Conforming applications are responsible for performing any cleanup actions needed for data
 53998 structures associated with the key to be deleted, including data referenced by thread-specific
 53999 data values. No such cleanup is done by *pthread_key_delete()*. In particular, destructor functions
 54000 are not called. There are several reasons for this division of responsibility:

- 54001 1. The associated destructor functions used to free thread-specific data at thread exit time
 54002 are only guaranteed to work correctly when called in the thread that allocated the thread-
 54003 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot
 54004 be used to free thread-specific data in other threads at key deletion time. Attempting to
 54005 have them called by other threads at key deletion time would require other threads to be
 54006 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,
 54007 including holding locks necessary for the destructor to run, this approach would fail. In
 54008 general, there is no safe mechanism whereby an implementation could free thread-
 54009 specific data at key deletion time.
- 54010 2. Even if there were a means of safely freeing thread-specific data associated with keys to
 54011 be deleted, doing so would require that implementations be able to enumerate the
 54012 threads with non-NULL data and potentially keep them from creating more thread-

54013 specific data while the key deletion is occurring. This special case could cause extra
54014 synchronization in the normal case, which would otherwise be unnecessary.

54015 For an application to know that it is safe to delete a key, it has to know that all the threads that
54016 might potentially ever use the key do not attempt to use it again. For example, it could know
54017 this if all the client threads have called a cleanup procedure declaring that they are through with
54018 the module that is being shut down, perhaps by setting a reference count to zero.

54019 If an implementation detects that the value specified by the *key* argument to *pthread_key_delete()*
54020 does not refer to a key value obtained from *pthread_key_create()* or refers to a key that has been
54021 deleted with *pthread_key_delete()*, it is recommended that the function should fail and report an
54022 [EINVAL] error.

54023 FUTURE DIRECTIONS

54024 None.

54025 SEE ALSO

54026 [*pthread_key_create\(\)*](#)

54027 XBD <[pthread.h](#)>

54028 CHANGE HISTORY

54029 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54030 Issue 6

54031 The *pthread_key_delete()* function is marked as part of the Threads option.

54032 Issue 7

54033 The *pthread_key_delete()* function is moved from the Threads option to the Base.

54034 The [EINVAL] error for a key value not obtained from *pthread_key_create()* or a key deleted with
54035 *pthread_key_delete()* is removed; this condition results in undefined behavior.

54036 **NAME**

54037 pthread_kill — send a signal to a thread

54038 **SYNOPSIS**

```
54039 CX #include <signal.h>
54040 int pthread_kill(pthread_t thread, int sig);
```

54041 **DESCRIPTION**54042 The *pthread_kill()* function shall request that a signal be delivered to the specified thread.54043 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent.54044 **RETURN VALUE**54045 Upon successful completion, the function shall return a value of zero. Otherwise, the function
54046 shall return an error number. If the *pthread_kill()* function fails, no signal shall be sent.54047 **ERRORS**54048 The *pthread_kill()* function shall fail if:54049 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.54050 The *pthread_kill()* function shall not return an error code of [EINTR].54051 **EXAMPLES**

54052 None.

54053 **APPLICATION USAGE**54054 The *pthread_kill()* function provides a mechanism for asynchronously directing a signal at a
54055 thread in the calling process. This could be used, for example, by one thread to affect broadcast
54056 delivery of a signal to a set of threads.54057 Note that *pthread_kill()* only causes the signal to be handled in the context of the given thread;
54058 the signal action (termination or stopping) affects the process as a whole.54059 **RATIONALE**54060 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
54061 that the function should fail and report an [ESRCH] error.54062 Existing implementations vary on the result of a *pthread_kill()* with a thread ID indicating an
54063 inactive thread (a terminated thread that has not been detached or joined). Some indicate success
54064 on such a call, while others give an error of [ESRCH]. Since the definition of thread lifetime in
54065 this volume of POSIX.1-2017 covers inactive threads, the [ESRCH] error as described is
54066 inappropriate in this case. In particular, this means that an application cannot have one thread
54067 check for termination of another with *pthread_kill()*.54068 **FUTURE DIRECTIONS**54069 A future version of this standard may require that *pthread_kill()* not fail with [ESRCH] in the
54070 case of sending signals to an inactive thread (a terminated thread not yet detached or joined),
54071 even though no signal will be delivered because the thread is no longer running.54072 **SEE ALSO**54073 *kill()*, *pthread_self()*, *raise()*

54074 XBD <signal.h>

54075 CHANGE HISTORY

54076 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54077 Issue 6

54078 The *pthread_kill()* function is marked as part of the Threads option.

54079 The APPLICATION USAGE section is added.

54080 Issue 7

54081 The *pthread_kill()* function is moved from the Threads option to the Base.

54082 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

54083 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0277 [765] is applied.

54084 **NAME**

54085 pthread_mutex_consistent — mark state protected by robust mutex as consistent

54086 **SYNOPSIS**

54087 #include <pthread.h>

54088 int pthread_mutex_consistent(pthread_mutex_t *mutex);

54089 **DESCRIPTION**54090 If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent()* function can be
54091 used to mark the state protected by the mutex referenced by *mutex* as consistent again.54092 If an owner of a robust mutex terminates while holding the mutex, the mutex becomes
54093 inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the
54094 return value [EOWNERDEAD]. In this case, the mutex does not become normally usable again
54095 until the state is marked consistent.54096 If the thread which acquired the mutex lock with the return value [EOWNERDEAD] terminates
54097 before calling either *pthread_mutex_consistent()* or *pthread_mutex_unlock()*, the next thread that
54098 acquires the mutex lock shall be notified about the state of the mutex by the return value
54099 [EOWNERDEAD].54100 The behavior is undefined if the value specified by the *mutex* argument to
54101 *pthread_mutex_consistent()* does not refer to an initialized mutex.54102 **RETURN VALUE**54103 Upon successful completion, the *pthread_mutex_consistent()* function shall return zero.

54104 Otherwise, an error value shall be returned to indicate the error.

54105 **ERRORS**54106 The *pthread_mutex_consistent()* function shall fail if:54107 [EINVAL] The mutex object referenced by *mutex* is not robust or does not protect an
54108 inconsistent state.

54109 These functions shall not return an error code of [EINTR].

54110 **EXAMPLES**

54111 None.

54112 **APPLICATION USAGE**54113 The *pthread_mutex_consistent()* function is only responsible for notifying the implementation that
54114 the state protected by the mutex has been recovered and that normal operations with the mutex
54115 can be resumed. It is the responsibility of the application to recover the state so it can be reused.
54116 If the application is not able to perform the recovery, it can notify the implementation that the
54117 situation is unrecoverable by a call to *pthread_mutex_unlock()* without a prior call to
54118 *pthread_mutex_consistent()*, in which case subsequent threads that attempt to lock the mutex will
54119 fail to acquire the lock and be returned [ENOTRECOVERABLE].54120 **RATIONALE**54121 If an implementation detects that the value specified by the *mutex* argument to
54122 *pthread_mutex_consistent()* does not refer to an initialized mutex, it is recommended that the
54123 function should fail and report an [EINVAL] error.54124 **FUTURE DIRECTIONS**

54125 None.

54126 **SEE ALSO**

54127 [pthread_mutex_lock\(\)](#), [pthread_mutexattr_getrobust\(\)](#)

54128 XBD [<pthread.h>](#)

54129 **CHANGE HISTORY**

54130 First released in Issue 7.

54131 **NAME**

54132 pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

54133 **SYNOPSIS**

```
54134 #include <pthread.h>
54135 int pthread_mutex_destroy(pthread_mutex_t *mutex);
54136 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
54137     const pthread_mutexattr_t *restrict attr);
54138 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

54139 **DESCRIPTION**

54140 The *pthread_mutex_destroy()* function shall destroy the mutex object referenced by *mutex*; the
 54141 mutex object becomes, in effect, uninitialized. An implementation may cause
 54142 *pthread_mutex_destroy()* to set the object referenced by *mutex* to an invalid value.

54143 A destroyed mutex object can be reinitialized using *pthread_mutex_init()*; the results of otherwise
 54144 referencing the object after it has been destroyed are undefined.

54145 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked
 54146 mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a
 54147 *pthread_cond_timedwait()* or *pthread_cond_wait()* call by another thread, results in undefined
 54148 behavior.

54149 The *pthread_mutex_init()* function shall initialize the mutex referenced by *mutex* with attributes
 54150 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the
 54151 same as passing the address of a default mutex attributes object. Upon successful initialization,
 54152 the state of the mutex becomes initialized and unlocked.

54153 See [Section 2.9.9](#) (on page 523) for further requirements.

54154 Attempting to initialize an already initialized mutex results in undefined behavior.

54155 In cases where default mutex attributes are appropriate, the macro
 54156 PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes. The effect shall be
 54157 equivalent to dynamic initialization by a call to *pthread_mutex_init()* with parameter *attr*
 54158 specified as NULL, except that no error checks are performed.

54159 The behavior is undefined if the value specified by the *mutex* argument to
 54160 *pthread_mutex_destroy()* does not refer to an initialized mutex.

54161 The behavior is undefined if the value specified by the *attr* argument to *pthread_mutex_init()*
 54162 does not refer to an initialized mutex attributes object.

54163 **RETURN VALUE**

54164 If successful, the *pthread_mutex_destroy()* and *pthread_mutex_init()* functions shall return zero;
 54165 otherwise, an error number shall be returned to indicate the error.

54166 **ERRORS**

54167 The *pthread_mutex_init()* function shall fail if:

54168 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 54169 another mutex.

54170 [ENOMEM] Insufficient memory exists to initialize the mutex.

54171 [EPERM] The caller does not have the privilege to perform the operation.

54172 The *pthread_mutex_init()* function may fail if:
 54173 [EINVAL] The attributes object referenced by *attr* has the robust mutex attribute set
 54174 without the process-shared attribute being set.
 54175 These functions shall not return an error code of [EINTR].

54176 EXAMPLES

54177 None.

54178 APPLICATION USAGE

54179 None.

54180 RATIONALE

54181 If an implementation detects that the value specified by the *mutex* argument to
 54182 *pthread_mutex_destroy()* does not refer to an initialized mutex, it is recommended that the
 54183 function should fail and report an [EINVAL] error.

54184 If an implementation detects that the value specified by the *mutex* argument to
 54185 *pthread_mutex_destroy()* or *pthread_mutex_init()* refers to a locked mutex or a mutex that is
 54186 referenced (for example, while being used in a *pthread_cond_timedwait()* or *pthread_cond_wait()*)
 54187 by another thread, or detects that the value specified by the *mutex* argument to
 54188 *pthread_mutex_init()* refers to an already initialized mutex, it is recommended that the function
 54189 should fail and report an [EBUSY] error.

54190 If an implementation detects that the value specified by the *attr* argument to
 54191 *pthread_mutex_init()* does not refer to an initialized mutex attributes object, it is recommended
 54192 that the function should fail and report an [EINVAL] error.

54193 Alternate Implementations Possible

54194 This volume of POSIX.1-2017 supports several alternative implementations of mutexes. An
 54195 implementation may store the lock directly in the object of type **pthread_mutex_t**. Alternatively,
 54196 an implementation may store the lock in the heap and merely store a pointer, handle, or unique
 54197 ID in the mutex object. Either implementation has advantages or may be required on certain
 54198 hardware configurations. So that portable code can be written that is invariant to this choice, this
 54199 volume of POSIX.1-2017 does not define assignment or equality for this type, and it uses the
 54200 term “initialize” to reinforce the (more restrictive) notion that the lock may actually reside in the
 54201 mutex object itself.

54202 Note that this precludes an over-specification of the type of the mutex or condition variable and
 54203 motivates the opaqueness of the type.

54204 An implementation is permitted, but not required, to have *pthread_mutex_destroy()* store an
 54205 illegal value into the mutex. This may help detect erroneous programs that try to lock (or
 54206 otherwise reference) a mutex that has already been destroyed.

54207 Tradeoff Between Error Checks and Performance Supported

54208 Many error conditions that can occur are not required to be detected by the implementation in
 54209 order to let implementations trade off performance *versus* degree of error checking according to
 54210 the needs of their specific applications and execution environment. As a general rule, conditions
 54211 caused by the system (such as insufficient memory) are required to be detected, but conditions
 54212 caused by an erroneously coded application (such as failing to provide adequate
 54213 synchronization to prevent a mutex from being deleted while in use) are specified to result in
 54214 undefined behavior.

54215 A wide range of implementations is thus made possible. For example, an implementation

54216 intended for application debugging may implement all of the error checks, but an
 54217 implementation running a single, provably correct application under very tight performance
 54218 constraints in an embedded computer might implement minimal checks. An implementation
 54219 might even be provided in two versions, similar to the options that compilers provide: a full-
 54220 checking, but slower version; and a limited-checking, but faster version. To forbid this
 54221 optionality would be a disservice to users.

54222 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly
 54223 coded) application might do, and by defining that resource-not-available errors are mandatory,
 54224 this volume of POSIX.1-2017 ensures that a fully-conforming application is portable across the
 54225 full range of implementations, while not forcing all implementations to add overhead to check
 54226 for numerous things that a correct program never does. When the behavior is undefined, no
 54227 error number is specified to be returned on implementations that do detect the condition. This is
 54228 because undefined behavior means *anything* can happen, which includes returning with any
 54229 value (which might happen to be a valid, but different, error number). However, since the error
 54230 number might be useful to application developers when diagnosing problems during
 54231 application development, a recommendation is made in rationale that implementors should
 54232 return a particular error number if their implementation does detect the condition.

54233 **Why No Limits are Defined**

54234 Defining symbols for the maximum number of mutexes and condition variables was considered
 54235 but rejected because the number of these objects may change dynamically. Furthermore, many
 54236 implementations place these objects into application memory; thus, there is no explicit
 54237 maximum.

54238 **Static Initializers for Mutexes and Condition Variables**

54239 Providing for static initialization of statically allocated synchronization objects allows modules
 54240 with private static synchronization variables to avoid runtime initialization tests and overhead.
 54241 Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in
 54242 C libraries, where for various reasons the design calls for self-initialization instead of requiring
 54243 an explicit module initialization function to be called. An example use of static initialization
 54244 follows.

54245 Without static initialization, a self-initializing routine *foo()* might look as follows:

```
54246 static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
54247 static pthread_mutex_t foo_mutex;

54248 void foo_init()
54249 {
54250     pthread_mutex_init(&foo_mutex, NULL);
54251 }

54252 void foo()
54253 {
54254     pthread_once(&foo_once, foo_init);
54255     pthread_mutex_lock(&foo_mutex);
54256     /* Do work. */
54257     pthread_mutex_unlock(&foo_mutex);
54258 }
```

54259 With static initialization, the same routine could be coded as follows:

```
54260 static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
54261 void foo()  
54262 {  
54263     pthread_mutex_lock(&foo_mutex);  
54264     /* Do work. */  
54265     pthread_mutex_unlock(&foo_mutex);  
54266 }
```

54267 Note that the static initialization both eliminates the need for the initialization test inside
54268 *pthread_once()* and the fetch of *&foo_mutex* to learn the address to be passed to
54269 *pthread_mutex_lock()* or *pthread_mutex_unlock()*.

54270 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a
54271 large class of systems; those where the (entire) synchronization object can be stored in
54272 application memory.

54273 Yet the locking performance question is likely to be raised for machines that require mutexes to
54274 be allocated out of special memory. Such machines actually have to have mutexes and possibly
54275 condition variables contain pointers to the actual hardware locks. For static initialization to work
54276 on such machines, *pthread_mutex_lock()* also has to test whether or not the pointer to the actual
54277 lock has been allocated. If it has not, *pthread_mutex_lock()* has to initialize it before use. The
54278 reservation of such resources can be made when the program is loaded, and hence return codes
54279 have not been added to mutex locking and condition variable waiting to indicate failure to
54280 complete initialization.

54281 This runtime test in *pthread_mutex_lock()* would at first seem to be extra work; an extra test is
54282 required to see whether the pointer has been initialized. On most machines this would actually
54283 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the
54284 pointer if it has already been initialized. While the test might seem to add extra work, the extra
54285 effort of testing a register is usually negligible since no extra memory references are actually
54286 done. As more and more machines provide caches, the real expenses are memory references, not
54287 instructions executed.

54288 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*
54289 overhead in the most important case: on the lock operations that occur *after* the lock has been
54290 initialized. This can be done by shifting more overhead to the less frequent operation:
54291 initialization. Since out-of-line mutex allocation also means that an address has to be
54292 dereferenced to find the actual lock, one technique that is widely applicable is to have static
54293 initialization store a bogus value for that address; in particular, an address that causes a machine
54294 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity
54295 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent
54296 lock operations incur no extra overhead since they do not “fault”. This is merely one technique
54297 that can be used to support static initialization, while not adversely affecting the performance of
54298 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

54299 The locking overhead for machines doing out-of-line mutex allocation is thus similar for
54300 modules being implicitly initialized, where it is improved for those doing mutex allocation
54301 entirely inline. The inline case is thus made much faster, and the out-of-line case is not
54302 significantly worse.

54303 Besides the issue of locking performance for such machines, a concern is raised that it is possible
54304 that threads would serialize contending for initialization locks when attempting to finish
54305 initializing statically allocated mutexes. (Such finishing would typically involve taking an
54306 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing
54307 the internal lock.) First, many implementations would reduce such serialization by hashing on
54308 the mutex address. Second, such serialization can only occur a bounded number of times. In

54309 particular, it can happen at most as many times as there are statically allocated synchronization
 54310 objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init()* or
 54311 *pthread_cond_init()*.

54312 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient
 54313 performance for an application on some implementation, the application can avoid static
 54314 initialization altogether by explicitly initializing all synchronization objects with the
 54315 corresponding *pthread_*_init()* functions, which are supported by all implementations. An
 54316 implementation can also document the tradeoffs and advise which initialization technique is
 54317 more efficient for that particular implementation.

54318 **Destroying Mutexes**

54319 A mutex can be destroyed immediately after it is unlocked. However, since attempting to
 54320 destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is
 54321 being used in a *pthread_cond_timedwait()* or *pthread_cond_wait()* call by another thread, results in
 54322 undefined behavior, care must be taken to ensure that no other thread may be referencing the
 54323 mutex.

54324 **Robust Mutexes**

54325 Implementations are required to provide robust mutexes for mutexes with the process-shared
 54326 attribute set to PTHREAD_PROCESS_SHARED. Implementations are allowed, but not required,
 54327 to provide robust mutexes when the process-shared attribute is set to
 54328 PTHREAD_PROCESS_PRIVATE.

54329 **FUTURE DIRECTIONS**

54330 None.

54331 **SEE ALSO**

54332 *pthread_mutex_getprioceiling()*, *pthread_mutexattr_getrobust()*, *pthread_mutex_lock()*,
 54333 *pthread_mutex_timedlock()*, *pthread_mutexattr_getpshared()*

54334 XBD <[pthread.h](#)>

54335 **CHANGE HISTORY**

54336 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54337 **Issue 6**

54338 The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are marked as part of the
 54339 Threads option.

54340 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
 54341 IEEE Std 1003.1d-1999.

54342 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

54343 The **restrict** keyword is added to the *pthread_mutex_init()* prototype for alignment with the
 54344 ISO/IEC 9899: 1999 standard.

54345 **Issue 7**

54346 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

54347 The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are moved from the Threads
 54348 option to the Base.

54349 The [EINVAL] error for an uninitialized mutex or an uninitialized mutex attributes object is
 54350 removed; this condition results in undefined behavior.

54351 The [EBUSY] error for a locked mutex, a mutex that is referenced, or an already initialized mutex
54352 is removed; this condition results in undefined behavior.

54353 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0460 [70,428] is applied.

54354 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0278 [811], XSH/TC2-2008/0279 [972],
54355 and XSH/TC2-2008/0280 [811] are applied.

54356 **NAME**

54357 pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling
 54358 of a mutex (**REALTIME THREADS**)

54359 **SYNOPSIS**

```
54360 RPP|TPP #include <pthread.h>
54361 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
54362 int *restrict prioceiling);
54363 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
54364 int prioceiling, int *restrict old_ceiling);
```

54365 **DESCRIPTION**

54366 The *pthread_mutex_getprioceiling()* function shall return the current priority ceiling of the mutex.

54367 The *pthread_mutex_setprioceiling()* function shall attempt to lock the mutex as if by a call to
 54368 *pthread_mutex_lock()*, except that the process of locking the mutex need not adhere to the priority
 54369 protect protocol. On acquiring the mutex it shall change the mutex's priority ceiling and then
 54370 release the mutex as if by a call to *pthread_mutex_unlock()*. When the change is successful, the
 54371 previous value of the priority ceiling shall be returned in *old_ceiling*.

54372 If the *pthread_mutex_setprioceiling()* function fails, the mutex priority ceiling shall not be
 54373 changed.

54374 **RETURN VALUE**

54375 If successful, the *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions shall
 54376 return zero; otherwise, an error number shall be returned to indicate the error.

54377 **ERRORS**

54378 These functions shall fail if:

- 54379 [EINVAL] The protocol attribute of *mutex* is PTHREAD_PRIO_NONE.
- 54380 [EPERM] The implementation requires appropriate privileges to perform the operation
 54381 and the caller does not have appropriate privileges.

54382 The *pthread_mutex_setprioceiling()* function shall fail if:

- 54383 [EAGAIN] The mutex could not be acquired because the maximum number of recursive
 54384 locks for *mutex* has been exceeded.
- 54385 [EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current
 54386 thread already owns the mutex.
- 54387 [EINVAL] The mutex was created with the protocol attribute having the value
 54388 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
 54389 the mutex's current priority ceiling, and the implementation adheres to the
 54390 priority protect protocol in the process of locking the mutex.
- 54391 [ENOTRECOVERABLE]
 54392 The mutex is a robust mutex and the state protected by the mutex is not
 54393 recoverable.
- 54394 [EOWNERDEAD]
 54395 The mutex is a robust mutex and the process containing the previous owning
 54396 thread terminated while holding the mutex lock. The mutex lock shall be
 54397 acquired by the calling thread and it is up to the new owner to make the state
 54398 consistent (see *pthread_mutex_lock()*).

54399 The *pthread_mutex_setprioceiling()* function may fail if:

54400 [EDEADLK] A deadlock condition was detected.

54401 [EINVAL] The priority requested by *prioceiling* is out of range.

54402 [EOWNERDEAD]

54403 The mutex is a robust mutex and the previous owning thread terminated

54404 while holding the mutex lock. The mutex lock shall be acquired by the calling

54405 thread and it is up to the new owner to make the state consistent (see

54406 *pthread_mutex_lock()*).

54407 These functions shall not return an error code of [EINTR].

EXAMPLES

54408 None.

APPLICATION USAGE

54411 None.

RATIONALE

54413 None.

FUTURE DIRECTIONS

54415 None.

SEE ALSO

54417 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *pthread_mutex_timedlock()*

54418 XBD <pthread.h>

CHANGE HISTORY

54420 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54421 Marked as part of the Realtime Threads Feature Group.

Issue 6

54423 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are marked as

54424 part of the Threads and Thread Priority Protection options.

54425 The [ENOSYS] error conditions have been removed.

54426 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with

54427 IEEE Std 1003.1d-1999.

54428 The **restrict** keyword is added to the *pthread_mutex_getprioceiling()* and

54429 *pthread_mutex_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

54431 SD5-XSH-ERN-39 is applied.

54432 Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a ``may fail''

54433 error.

54434 SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a ``shall fail'' error case

54435 for when the protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

54436 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are moved from

54437 the Threads option to require support of either the Robust Mutex Priority Protection option or

54438 the Non-Robust Mutex Priority Protection option.

54439
54440

The DESCRIPTION and ERRORS sections are updated to account properly for all of the various mutex types.

54441 **NAME**

54442 pthread_mutex_init — destroy and initialize a mutex

54443 **SYNOPSIS**

54444 #include <pthread.h>

54445 int pthread_mutex_init(pthread_mutex_t *restrict mutex,

54446 const pthread_mutexattr_t *restrict attr);

54447 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

54448 **DESCRIPTION**

54449 Refer to [pthread_mutex_destroy\(\)](#).

54450 **NAME**

54451 pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a
54452 mutex

54453 **SYNOPSIS**

```
54454 #include <pthread.h>
54455 int pthread_mutex_lock(pthread_mutex_t *mutex);
54456 int pthread_mutex_trylock(pthread_mutex_t *mutex);
54457 int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

54458 **DESCRIPTION**

54459 The mutex object referenced by *mutex* shall be locked by a call to *pthread_mutex_lock()* that
54460 returns zero or [EOWNERDEAD]. If the mutex is already locked by another thread, the calling
54461 thread shall block until the mutex becomes available. This operation shall return with the mutex
54462 object referenced by *mutex* in the locked state with the calling thread as its owner. If a thread
54463 attempts to relock a mutex that it has already locked, *pthread_mutex_lock()* shall behave as
54464 described in the **Relock** column of the following table. If a thread attempts to unlock a mutex
54465 that it has not locked or a mutex which is unlocked, *pthread_mutex_unlock()* shall behave as
54466 described in the **Unlock When Not Owner** column of the following table.

54467	Mutex Type	Robustness	Relock	Unlock When Not Owner
54468	NORMAL	non-robust	deadlock	undefined behavior
54469	NORMAL	robust	deadlock	error returned
54470	ERRORCHECK	either	error returned	error returned
54471	RECURSIVE	either	recursive (see below)	error returned
54472	DEFAULT	non-robust	undefined behavior •	undefined behavior •
54473	DEFAULT	robust	undefined behavior •	error returned
54474				
54475				
54476				

54477 •the mutex type is PTHREAD_MUTEX_DEFAULT, the behavior of *pthread_mutex_lock()*
54478 may correspond to one of the three other standard mutex types as described in the table
54479 above. If it does not correspond to one of those three, the behavior is undefined for the cases
54480 marked †.

54481 Where the table indicates recursive behavior, the mutex shall maintain the concept of a lock
54482 count. When a thread successfully acquires a mutex for the first time, the lock count shall be set
54483 to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each
54484 time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock
54485 count reaches zero, the mutex shall become available for other threads to acquire.

54486 The *pthread_mutex_trylock()* function shall be equivalent to *pthread_mutex_lock()*, except that if
54487 the mutex object referenced by *mutex* is currently locked (by any thread, including the current
54488 thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE
54489 and the mutex is currently owned by the calling thread, the mutex lock count shall be
54490 incremented by one and the *pthread_mutex_trylock()* function shall immediately return success.

54491 The *pthread_mutex_unlock()* function shall release the mutex object referenced by *mutex*. The
54492 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are
54493 threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock()* is called,
54494 resulting in the mutex becoming available, the scheduling policy shall determine which thread
54495 shall acquire the mutex.

54496 (In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available
54497 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

54498 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the
54499 thread shall resume waiting for the mutex as if it was not interrupted.

54500 If *mutex* is a robust mutex and the process containing the owning thread terminated while
54501 holding the mutex lock, a call to *pthread_mutex_lock()* shall return the error value
54502 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding
54503 the mutex lock, a call to *pthread_mutex_lock()* may return the error value [EOWNERDEAD] even
54504 if the process in which the owning thread resides has not terminated. In these cases, the mutex is
54505 locked by the thread but the state it protects is marked as inconsistent. The application should
54506 ensure that the state is made consistent for reuse and when that is complete call
54507 *pthread_mutex_consistent()*. If the application is unable to recover the state, it should unlock the
54508 mutex without a prior call to *pthread_mutex_consistent()*, after which the mutex is marked
54509 permanently unusable.

54510 If *mutex* does not refer to an initialized mutex object, the behavior of *pthread_mutex_lock()*,
54511 *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* is undefined.

54512 RETURN VALUE

54513 If successful, the *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()*
54514 functions shall return zero; otherwise, an error number shall be returned to indicate the error.

54515 ERRORS

54516 The *pthread_mutex_lock()* and *pthread_mutex_trylock()* functions shall fail if:

54517 [EAGAIN] The mutex could not be acquired because the maximum number of recursive
54518 locks for *mutex* has been exceeded.

54519 RPP|TPP [EINVAL] The *mutex* was created with the protocol attribute having the value
54520 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
54521 the mutex's current priority ceiling.

54522 [ENOTRECOVERABLE]
54523 The state protected by the mutex is not recoverable.

54524 [EOWNERDEAD]
54525 The mutex is a robust mutex and the process containing the previous owning
54526 thread terminated while holding the mutex lock. The mutex lock shall be
54527 acquired by the calling thread and it is up to the new owner to make the state
54528 consistent.

54529 The *pthread_mutex_lock()* function shall fail if:

54530 [EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current
54531 thread already owns the mutex.

54532 The *pthread_mutex_trylock()* function shall fail if:

54533 [EBUSY] The *mutex* could not be acquired because it was already locked.

54534 The *pthread_mutex_unlock()* function shall fail if:

54535 [EPERM] The mutex type is PTHREAD_MUTEX_ERRORCHECK or
54536 PTHREAD_MUTEX_RECURSIVE, or the mutex is a robust mutex, and the
54537 current thread does not own the mutex.

54538 The `pthread_mutex_lock()` and `pthread_mutex_trylock()` functions may fail if:
 54539 [EOWNERDEAD]
 54540 The mutex is a robust mutex and the previous owning thread terminated
 54541 while holding the mutex lock. The mutex lock shall be acquired by the calling
 54542 thread and it is up to the new owner to make the state consistent.

54543 The `pthread_mutex_lock()` function may fail if:
 54544 [EDEADLK] A deadlock condition was detected.
 54545 These functions shall not return an error code of [EINTR].

54546 EXAMPLES

54547 None.

54548 APPLICATION USAGE

54549 Applications that have assumed that non-zero return values are errors will need updating for
 54550 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting
 54551 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error
 54552 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If
 54553 an application is supposed to work with normal and robust mutexes it should check all return
 54554 values for error conditions and if necessary take appropriate action.

54555 RATIONALE

54556 Mutex objects are intended to serve as a low-level primitive from which other thread
 54557 synchronization functions can be built. As such, the implementation of mutexes should be as
 54558 efficient as possible, and this has ramifications on the features available at the interface.

54559 The mutex functions and the particular default settings of the mutex attributes have been
 54560 motivated by the desire to not preclude fast, inlined implementations of mutex locking and
 54561 unlocking.

54562 Since most attributes only need to be checked when a thread is going to be blocked, the use of
 54563 attributes does not slow the (common) mutex-locking case.

54564 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it
 54565 would require storing the current thread ID when each mutex is locked, and this could incur
 54566 unacceptable levels of overhead. Similar arguments apply to a `mutex_tryunlock` operation.

54567 For further rationale on the extended mutex types, see XRAT [Threads Extensions](#) (on page 3642).

54568 If an implementation detects that the value specified by the `mutex` argument does not refer to an
 54569 initialized mutex object, it is recommended that the function should fail and report an [EINVAL]
 54570 error.

54571 FUTURE DIRECTIONS

54572 None.

54573 SEE ALSO

54574 [pthread_mutex_consistent\(\)](#), [pthread_mutex_destroy\(\)](#), [pthread_mutex_timedlock\(\)](#),
 54575 [pthread_mutexattr_getrobust\(\)](#)

54576 XBD [Section 4.12](#) (on page 111), [<pthread.h>](#)

54577 CHANGE HISTORY

54578 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54579 **Issue 6**

54580 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are
54581 marked as part of the Threads option.

54582 The following new requirements on POSIX implementations derive from alignment with the
54583 Single UNIX Specification:

54584 The behavior when attempting to relock a mutex is defined.

54585 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
54586 IEEE Std 1003.1d-1999.

54587 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/98 is applied, updating the ERRORS
54588 section so that the [EDEADLK] error includes detection of a deadlock condition. The
54589 RATIONALE section is also reworded to take into account non-XSI-conformant systems.

54590 **Issue 7**

54591 SD5-XSH-ERN-43 is applied, marking the “shall fail” case of the [EINVAL] error as dependent
54592 on the Thread Priority Protection option.

54593 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

54594 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are
54595 moved from the Threads option to the Base.

54596 The following extended mutex types are moved from the XSI option to the Base:

54597 PTHREAD_MUTEX_NORMAL
54598 PTHREAD_MUTEX_ERRORCHECK
54599 PTHREAD_MUTEX_RECURSIVE
54600 PTHREAD_MUTEX_DEFAULT

54601 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized
54602 mutex.

54603 The ERRORS section is updated to account properly for all of the various mutex types.

54604 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0461 [121], XSH/TC1-2008/0462
54605 [92,428], and XSH/TC1-2008/0463 [121] are applied.

54606 **NAME**

54607 pthread_mutex_setprioceiling ¶ change the priority ceiling of a mutex **REALTIME**
54608 **THREADS**)

54609 **SYNOPSIS**

```
54610 RPP|TPP #include <pthread.h>  
54611 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,  
54612 int prioceiling, int *restrict old_ceiling);
```

54613 **DESCRIPTION**

54614 Refer to [pthread_mutex_getprioceiling\(\)](#).

54615 **NAME**

54616 pthread_mutex_timedlock — lock a mutex

54617 **SYNOPSIS**

```
54618 #include <pthread.h>
54619 #include <time.h>
54620 int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
54621     const struct timespec *restrict abstime);
```

54622 **DESCRIPTION**

54623 The *pthread_mutex_timedlock()* function shall lock the mutex object referenced by *mutex*. If the
 54624 mutex is already locked, the calling thread shall block until the mutex becomes available as in
 54625 the *pthread_mutex_lock()* function. If the mutex cannot be locked without waiting for another
 54626 thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

54627 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
 54628 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 54629 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
 54630 call.

54631 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 54632 be the resolution of the clock on which it is based. The **timespec** data type is defined in the
 54633 **<time.h>** header.

54634 Under no circumstance shall the function fail with a timeout if the mutex can be locked
 54635 immediately. The validity of the *abstime* parameter need not be checked if the mutex can be
 54636 locked immediately.

54637 RPI | TPI As a consequence of the priority inheritance rules (for mutexes initialized with the
 54638 PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the
 54639 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this
 54640 thread is no longer among the threads waiting for the mutex.

54641 If *mutex* is a robust mutex and the process containing the owning thread terminated while
 54642 holding the mutex lock, a call to *pthread_mutex_timedlock()* shall return the error value
 54643 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding
 54644 the mutex lock, a call to *pthread_mutex_timedlock()* may return the error value [EOWNERDEAD]
 54645 even if the process in which the owning thread resides has not terminated. In these cases, the
 54646 mutex is locked by the thread but the state it protects is marked as inconsistent. The application
 54647 should ensure that the state is made consistent for reuse and when that is complete call
 54648 *pthread_mutex_consistent()*. If the application is unable to recover the state, it should unlock the
 54649 mutex without a prior call to *pthread_mutex_consistent()*, after which the mutex is marked
 54650 permanently unusable.

54651 If *mutex* does not refer to an initialized mutex object, the behavior is undefined.

54652 **RETURN VALUE**

54653 If successful, the *pthread_mutex_timedlock()* function shall return zero; otherwise, an error
 54654 number shall be returned to indicate the error.

54655 **ERRORS**

54656 The *pthread_mutex_timedlock()* function shall fail if:

54657 [EAGAIN] The mutex could not be acquired because the maximum number of recursive
 54658 locks for *mutex* has been exceeded.

- 54659 [EDEADLK] The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current
54660 thread already owns the mutex.
- 54661 [EINVAL] The mutex was created with the protocol attribute having the value
54662 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
54663 the mutex' current priority ceiling.
- 54664 [EINVAL] The process or thread would have blocked, and the *abstime* parameter
54665 specified a nanoseconds field value less than zero or greater than or equal to
54666 1 000 million.
- 54667 [ENOTRECOVERABLE]
54668 The state protected by the mutex is not recoverable.
- 54669 [EOWNERDEAD]
54670 The mutex is a robust mutex and the process containing the previous owning
54671 thread terminated while holding the mutex lock. The mutex lock shall be
54672 acquired by the calling thread and it is up to the new owner to make the state
54673 consistent.
- 54674 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.
- 54675 The *pthread_mutex_timedlock()* function may fail if:
- 54676 [EDEADLK] A deadlock condition was detected.
- 54677 [EOWNERDEAD]
54678 The mutex is a robust mutex and the previous owning thread terminated
54679 while holding the mutex lock. The mutex lock shall be acquired by the calling
54680 thread and it is up to the new owner to make the state consistent.
- 54681 This function shall not return an error code of [EINTR].

54682 EXAMPLES

54683 None.

54684 APPLICATION USAGE

54685 Applications that have assumed that non-zero return values are errors will need updating for
54686 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting
54687 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error
54688 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If
54689 an application is supposed to work with normal and robust mutexes, it should check all return
54690 values for error conditions and if necessary take appropriate action.

54691 RATIONALE

54692 Refer to *pthread_mutex_lock()*.

54693 FUTURE DIRECTIONS

54694 None.

54695 SEE ALSO

54696 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *time()*

54697 XBD Section 4.12 (on page 111), [<pthread.h>](#), [<time.h>](#)

54698 CHANGE HISTORY

54699 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

54700 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/99 is applied, marking the last paragraph
54701 in the DESCRIPTION as part of the Thread Priority Inheritance option.

54702 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/100 is applied, updating the ERRORS
54703 section so that the [EDEADLK] error includes detection of a deadlock condition.

54704 **Issue 7**

54705 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

54706 The *pthread_mutex_timedlock()* function is moved from the Timeouts option to the Base.

54707 Functionality relating to the Timers option is moved to the Base.

54708 The DESCRIPTION is updated to clarify the behavior when *mutex* does not refer to an initialized
54709 mutex.

54710 The ERRORS section is updated to account properly for all of the various mutex types.

54711 **NAME**

54712 pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

54713 **SYNOPSIS**

54714 #include <pthread.h>

54715 int pthread_mutex_trylock(pthread_mutex_t *mutex);

54716 int pthread_mutex_unlock(pthread_mutex_t *mutex);

54717 **DESCRIPTION**54718 Refer to *pthread_mutex_lock()*.

54719 **NAME**

54720 pthread_mutexattr_destroy, pthread_mutexattr_init ‡ destroy and initialize the mutex
54721 attributes object

54722 **SYNOPSIS**

```
54723 #include <pthread.h>
54724 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
54725 int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

54726 **DESCRIPTION**

54727 The *pthread_mutexattr_destroy()* function shall destroy a mutex attributes object; the object
54728 becomes, in effect, uninitialized. An implementation may cause *pthread_mutexattr_destroy()* to
54729 set the object referenced by *attr* to an invalid value.

54730 A destroyed *attr* attributes object can be reinitialized using *pthread_mutexattr_init()*; the results of
54731 otherwise referencing the object after it has been destroyed are undefined.

54732 The *pthread_mutexattr_init()* function shall initialize a mutex attributes object *attr* with the
54733 default value for all of the attributes defined by the implementation.

54734 Results are undefined if *pthread_mutexattr_init()* is called specifying an already initialized *attr*
54735 attributes object.

54736 After a mutex attributes object has been used to initialize one or more mutexes, any function
54737 affecting the attributes object (including destruction) shall not affect any previously initialized
54738 mutexes.

54739 The behavior is undefined if the value specified by the *attr* argument to
54740 *pthread_mutexattr_destroy()* does not refer to an initialized mutex attributes object.

54741 **RETURN VALUE**

54742 Upon successful completion, *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* shall
54743 return zero; otherwise, an error number shall be returned to indicate the error.

54744 **ERRORS**

54745 The *pthread_mutexattr_init()* function shall fail if:

54746 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

54747 These functions shall not return an error code of [EINTR].

54748 **EXAMPLES**

54749 None.

54750 **APPLICATION USAGE**

54751 None.

54752 **RATIONALE**

54753 If an implementation detects that the value specified by the *attr* argument to
54754 *pthread_mutexattr_destroy()* does not refer to an initialized mutex attributes object, it is
54755 recommended that the function should fail and report an [EINVAL] error.

54756 See *pthread_attr_destroy()* for a general explanation of attributes. Attributes objects allow
54757 implementations to experiment with useful extensions and permit extension of this volume of
54758 POSIX.1-2017 without changing the existing functions. Thus, they provide for future
54759 extensibility of this volume of POSIX.1-2017 and reduce the temptation to standardize
54760 prematurely on semantics that are not yet widely implemented or understood.

54761 Examples of possible additional mutex attributes that have been discussed are *spin_only*,
54762 *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:

54763 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes
 54764 would transparently keep records of queue length, wait time, and so on.) Since there is not yet
 54765 wide agreement on the usefulness of these resulting from shared implementation and usage
 54766 experience, they are not yet specified in this volume of POSIX.1-2017. Mutex attributes objects,
 54767 however, make it possible to test out these concepts for possible standardization at a later time.

54768 **Mutex Attributes and Performance**

54769 Care has been taken to ensure that the default values of the mutex attributes have been defined
 54770 such that mutexes initialized with the defaults have simple enough semantics so that the locking
 54771 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few
 54772 other basic instructions).

54773 There is at least one implementation method that can be used to reduce the cost of testing at
 54774 lock-time if a mutex has non-default attributes. One such method that an implementation can
 54775 employ (and this can be made fully transparent to fully conforming POSIX applications) is to
 54776 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to
 54777 lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were
 54778 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-
 54779 default mutex. The underlying unlock operation is more complicated since the implementation
 54780 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending
 54781 on the hardware, there may be certain optimizations that can be used so that whatever mutex
 54782 attributes are considered “most frequently used” can be processed most efficiently.

54783 **Process Shared Memory and Synchronization**

54784 The existence of memory mapping functions in this volume of POSIX.1-2017 leads to the
 54785 possibility that an application may allocate the synchronization objects from this section in
 54786 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

54787 In order to permit such usage, while at the same time keeping the usual case (that is, usage
 54788 within a single process) efficient, a *process-shared* option has been defined.

54789 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the
 54790 *process-shared* attribute can be used to indicate that mutexes or condition variables may be
 54791 accessed by threads of multiple processes.

54792 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*
 54793 attribute so that the most efficient forms of these synchronization objects are created by default.

54794 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*
 54795 *shared* attribute may only be operated on by threads in the process that initialized them.
 54796 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*
 54797 *shared* attribute may be operated on by any thread in any process that has access to it. In
 54798 particular, these processes may exist beyond the lifetime of the initializing process. For example,
 54799 the following code implements a simple counting semaphore in a mapped file that may be used
 54800 by many processes.

```
54801 /* sem.h */
54802 struct semaphore {
54803     pthread_mutex_t lock;
54804     pthread_cond_t nonzero;
54805     unsigned count;
54806 };
54807 typedef struct semaphore semaphore_t;
```



```
54808 semaphore_t *semaphore_create(char *semaphore_name);
54809 semaphore_t *semaphore_open(char *semaphore_name);
54810 void semaphore_post(semaphore_t *semap);
54811 void semaphore_wait(semaphore_t *semap);
54812 void semaphore_close(semaphore_t *semap);

54813 /* sem.c */
54814 #include <sys/types.h>
54815 #include <sys/stat.h>
54816 #include <sys/mman.h>
54817 #include <fcntl.h>
54818 #include <pthread.h>
54819 #include "sem.h"

54820 semaphore_t *
54821 semaphore_create(char *semaphore_name)
54822 {
54823     int fd;
54824     semaphore_t *semap;
54825     pthread_mutexattr_t psharedm;
54826     pthread_condattr_t psharedc;

54827     fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
54828     if (fd < 0)
54829         return (NULL);
54830     (void) ftruncate(fd, sizeof(semaphore_t));
54831     (void) pthread_mutexattr_init(&psharedm);
54832     (void) pthread_mutexattr_setpshared(&psharedm,
54833         PTHREAD_PROCESS_SHARED);
54834     (void) pthread_condattr_init(&psharedc);
54835     (void) pthread_condattr_setpshared(&psharedc,
54836         PTHREAD_PROCESS_SHARED);
54837     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
54838         PROT_READ | PROT_WRITE, MAP_SHARED,
54839         fd, 0);
54840     close (fd);
54841     (void) pthread_mutex_init(&semap->lock, &psharedm);
54842     (void) pthread_cond_init(&semap->nonzero, &psharedc);
54843     semap->count = 0;
54844     return (semap);
54845 }

54846 semaphore_t *
54847 semaphore_open(char *semaphore_name)
54848 {
54849     int fd;
54850     semaphore_t *semap;

54851     fd = open(semaphore_name, O_RDWR, 0666);
54852     if (fd < 0)
54853         return (NULL);
54854     semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
54855         PROT_READ | PROT_WRITE, MAP_SHARED,
54856         fd, 0);
```

```

54857         close (fd);
54858         return (semap);
54859     }

54860     void
54861     semaphore_post(semaphore_t *semap)
54862     {
54863         pthread_mutex_lock(&semap->lock);
54864         if (semap->count == 0)
54865             pthread_cond_signal(&semap->nonzero);
54866         semap->count++;
54867         pthread_mutex_unlock(&semap->lock);
54868     }

54869     void
54870     semaphore_wait(semaphore_t *semap)
54871     {
54872         pthread_mutex_lock(&semap->lock);
54873         while (semap->count == 0)
54874             pthread_cond_wait(&semap->nonzero, &semap->lock);
54875         semap->count--;
54876         pthread_mutex_unlock(&semap->lock);
54877     }

54878     void
54879     semaphore_close(semaphore_t *semap)
54880     {
54881         munmap((void *) semap, sizeof(semaphore_t));
54882     }

```

54883 The following code is for three separate processes that create, post, and wait on a semaphore in
54884 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and
54885 decrement the counting semaphore (waiting and waking as required) even though they did not
54886 initialize the semaphore.

```

54887     /* create.c */
54888     #include "pthread.h"
54889     #include "sem.h"

54890     int
54891     main()
54892     {
54893         semaphore_t *semap;

54894         semap = semaphore_create("/tmp/semaphore");
54895         if (semap == NULL)
54896             exit(1);
54897         semaphore_close(semap);
54898         return (0);
54899     }

54900     /* post */
54901     #include "pthread.h"
54902     #include "sem.h"

54903     int

```

```

54904     main()
54905     {
54906         semaphore_t *semap;
54907
54908         semap = semaphore_open("/tmp/semaphore");
54909         if (semap == NULL)
54910             exit(1);
54911         semaphore_post(semap);
54912         semaphore_close(semap);
54913         return (0);
54914     }
54915
54916     /* wait */
54917     #include "pthread.h"
54918     #include "sem.h"
54919
54920     int
54921     main()
54922     {
54923         semaphore_t *semap;
54924
54925         semap = semaphore_open("/tmp/semaphore");
54926         if (semap == NULL)
54927             exit(1);
54928         semaphore_wait(semap);
54929         semaphore_close(semap);
54930         return (0);
54931     }

```

FUTURE DIRECTIONS

None.

SEE ALSO

[pthread_cond_destroy\(\)](#), [pthread_create\(\)](#), [pthread_mutex_destroy\(\)](#)

XBD [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The [pthread_mutexattr_destroy\(\)](#) and [pthread_mutexattr_init\(\)](#) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

Issue 7

The [pthread_mutexattr_destroy\(\)](#) and [pthread_mutexattr_init\(\)](#) functions are moved from the Threads option to the Base.

The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition results in undefined behavior.

54944 **NAME**

54945 pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling ‡ get and set the
 54946 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

54947 **SYNOPSIS**

```
54948 RPP|TPP #include <pthread.h>
54949
54949 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
54950     *restrict attr, int *restrict prioceiling);
54951 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
54952     int prioceiling);
```

54953 **DESCRIPTION**

54954 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions,
 54955 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to
 54956 by *attr* which was previously created by the function *pthread_mutexattr_init()*.

54957 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of
 54958 *prioceiling* are within the maximum range of priorities defined by **SCHED_FIFO**.

54959 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum
 54960 priority level at which the critical section guarded by the mutex is executed. In order to avoid
 54961 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal
 54962 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are
 54963 within the maximum range of priorities defined under the **SCHED_FIFO** scheduling policy.

54964 The behavior is undefined if the value specified by the *attr* argument to
 54965 *pthread_mutexattr_getprioceiling()* or *pthread_mutexattr_setprioceiling()* does not refer to an
 54966 initialized mutex attributes object.

54967 **RETURN VALUE**

54968 Upon successful completion, the *pthread_mutexattr_getprioceiling()* and
 54969 *pthread_mutexattr_setprioceiling()* functions shall return zero; otherwise, an error number shall be
 54970 returned to indicate the error.

54971 **ERRORS**

54972 These functions may fail if:

54973 [EINVAL] The value specified by *prioceiling* is invalid.

54974 [EPERM] The caller does not have the privilege to perform the operation.

54975 These functions shall not return an error code of [EINTR].

54976 **EXAMPLES**

54977 None.

54978 **APPLICATION USAGE**

54979 None.

54980 **RATIONALE**

54981 If an implementation detects that the value specified by the *attr* argument to
 54982 *pthread_mutexattr_getprioceiling()* or *pthread_mutexattr_setprioceiling()* does not refer to an
 54983 initialized mutex attributes object, it is recommended that the function should fail and report an
 54984 [EINVAL] error.

54985 **FUTURE DIRECTIONS**

54986 None.

54987 **SEE ALSO**54988 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*

54989 XBD <pthread.h>

54990 **CHANGE HISTORY**

54991 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

54992 Marked as part of the Realtime Threads Feature Group.

54993 **Issue 6**54994 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are marked
54995 as part of the Threads and Thread Priority Protection options.54996 The [ENOSYS] error condition has been removed as stubs need not be provided if an
54997 implementation does not support the Thread Priority Protection option.54998 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*
54999 argument.55000 The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling()* prototype for alignment
55001 with the ISO/IEC 9899:1999 standard.55002 **Issue 7**55003 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are moved
55004 from the Threads option to require support of either the Robust Mutex Priority Protection option
55005 or the Non-Robust Mutex Priority Protection option.55006 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
55007 results in undefined behavior.

55008 **NAME**

55009 pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol ‡ get and set the protocol
 55010 attribute of the mutex attributes object (**REALTIME THREADS**)

55011 **SYNOPSIS**

```
55012 MC1 #include <pthread.h>
55013
55014 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
55015 *restrict attr, int *restrict protocol);
55016 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
55017 int protocol);
```

55017 **DESCRIPTION**

55018 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions, respectively,
 55019 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was
 55020 previously created by the function *pthread_mutexattr_init()*.

55021 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of
 55022 *protocol* may be one of:

```
55023 RPI | TPI PTHREAD_PRIO_INHERIT
55024 MC1 PTHREAD_PRIO_NONE
55025 RPP | TPP PTHREAD_PRIO_PROTECT
```

55026 which are defined in the **<pthread.h>** header. The default value of the attribute shall be
 55027 PTHREAD_PRIO_NONE.

55028 When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority
 55029 and scheduling shall not be affected by its mutex ownership.

55030 RPI When a thread is blocking higher priority threads because of owning one or more robust
 55031 mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of
 55032 its priority or the priority of the highest priority thread waiting on any of the robust mutexes
 55033 owned by this thread and initialized with this protocol.

55034 TPI When a thread is blocking higher priority threads because of owning one or more non-robust
 55035 mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of
 55036 its priority or the priority of the highest priority thread waiting on any of the non-robust
 55037 mutexes owned by this thread and initialized with this protocol.

55038 RPP When a thread owns one or more robust mutexes initialized with the
 55039 PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest
 55040 of the priority ceilings of all the robust mutexes owned by this thread and initialized with this
 55041 attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

55042 TPP When a thread owns one or more non-robust mutexes initialized with the
 55043 PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest
 55044 of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with
 55045 this attribute, regardless of whether other threads are blocked on any of these non-robust
 55046 mutexes or not.

55047 While a thread is holding a mutex which has been initialized with the
 55048 PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be
 55049 subject to being moved to the tail of the scheduling queue at its priority in the event that its
 55050 original priority is changed, such as by a call to *sched_setparam()*. Likewise, when a thread

55051 unlocks a mutex that has been initialized with the PTHREAD_PRIO_INHERIT or
 55052 PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail
 55053 of the scheduling queue at its priority in the event that its original priority is changed.

55054 If a thread simultaneously owns several mutexes initialized with different protocols, it shall
 55055 execute at the highest of the priorities that it would have obtained by each of these protocols.

55056 RPI | TPI When a thread makes a call to *pthread_mutex_lock()*, the mutex was initialized with the protocol
 55057 attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked
 55058 because the mutex is owned by another thread, that owner thread shall inherit the priority level
 55059 of the calling thread as long as it continues to own the mutex. The implementation shall update
 55060 its execution priority to the maximum of its assigned priority and all its inherited priorities.
 55061 Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol*
 55062 attribute having the value PTHREAD_PRIO_INHERIT, the same priority inheritance effect shall
 55063 be propagated to this other owner thread, in a recursive manner.

55064 The behavior is undefined if the value specified by the *attr* argument to
 55065 *pthread_mutexattr_getprotocol()* or *pthread_mutexattr_setprotocol()* does not refer to an initialized
 55066 mutex attributes object.

55067 RETURN VALUE

55068 Upon successful completion, the *pthread_mutexattr_getprotocol()* and
 55069 *pthread_mutexattr_setprotocol()* functions shall return zero; otherwise, an error number shall be
 55070 returned to indicate the error.

55071 ERRORS

55072 The *pthread_mutexattr_setprotocol()* function shall fail if:

55073 [ENOTSUP] The value specified by *protocol* is an unsupported value.

55074 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions may fail if:

55075 [EINVAL] The value specified by *protocol* is invalid.

55076 [EPERM] The caller does not have the privilege to perform the operation.

55077 These functions shall not return an error code of [EINTR].

55078 EXAMPLES

55079 None.

55080 APPLICATION USAGE

55081 None.

55082 RATIONALE

55083 If an implementation detects that the value specified by the *attr* argument to
 55084 *pthread_mutexattr_getprotocol()* or *pthread_mutexattr_setprotocol()* does not refer to an initialized
 55085 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]
 55086 error.

55087 FUTURE DIRECTIONS

55088 None.

55089 SEE ALSO

55090 [*pthread_cond_destroy\(\)*](#), [*pthread_create\(\)*](#), [*pthread_mutex_destroy\(\)*](#)

55091 XBD <pthread.h>

55092 **CHANGE HISTORY**

55093 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55094 Marked as part of the Realtime Threads Feature Group.

55095 **Issue 6**

55096 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions are marked as
55097 part of the Threads option and either the Thread Priority Protection or Thread Priority
55098 Inheritance options.

55099 The [ENOSYS] error condition has been removed as stubs need not be provided if an
55100 implementation does not support the Thread Priority Protection or Thread Priority Inheritance
55101 options.

55102 The **restrict** keyword is added to the *pthread_mutexattr_getprotocol()* prototype for alignment
55103 with the ISO/IEC 9899:1999 standard.

55104 **Issue 7**

55105 SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the
55106 *protocol* attribute.

55107 SD5-XSH-ERN-188 is applied, updating the DESCRIPTION.

55108 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions are moved from
55109 the Threads option to require support of either the Non-Robust Mutex Priority Protection option
55110 or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection
55111 option or the Robust Mutex Priority Inheritance option.

55112 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
55113 results in undefined behavior.

55114 **NAME**

55115 pthread_mutexattr_getpshared, pthread_mutexattr_setpshared ‡get and set the process-shared
55116 attribute

55117 **SYNOPSIS**

```
55118 TSH #include <pthread.h>
55119
55119 int pthread_mutexattr_getpshared(const pthread_mutexattr_t
55120 *restrict attr, int *restrict pshared);
55121 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
55122 int pshared);
```

55123 **DESCRIPTION**

55124 The *pthread_mutexattr_getpshared()* function shall obtain the value of the *process-shared* attribute
55125 from the attributes object referenced by *attr*.

55126 The *pthread_mutexattr_setpshared()* function shall set the *process-shared* attribute in an initialized
55127 attributes object referenced by *attr*.

55128 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a mutex to be
55129 operated upon by any thread that has access to the memory where the mutex is allocated, even if
55130 the mutex is allocated in memory that is shared by multiple processes. See [Section 2.9.9](#) (on page
55131 523) for further requirements. The default value of the attribute shall be
55132 `PTHREAD_PROCESS_PRIVATE`.

55133 The behavior is undefined if the value specified by the *attr* argument to
55134 *pthread_mutexattr_getpshared()* or *pthread_mutexattr_setpshared()* does not refer to an initialized
55135 mutex attributes object.

55136 **RETURN VALUE**

55137 Upon successful completion, *pthread_mutexattr_setpshared()* shall return zero; otherwise, an error
55138 number shall be returned to indicate the error.

55139 Upon successful completion, *pthread_mutexattr_getpshared()* shall return zero and store the value
55140 of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.
55141 Otherwise, an error number shall be returned to indicate the error.

55142 **ERRORS**

55143 The *pthread_mutexattr_setpshared()* function may fail if:

55144 [EINVAL] The new value specified for the attribute is outside the range of legal values
55145 for that attribute.

55146 These functions shall not return an error code of [EINTR].

55147 **EXAMPLES**

55148 None.

55149 **APPLICATION USAGE**

55150 None.

55151 **RATIONALE**

55152 If an implementation detects that the value specified by the *attr* argument to
55153 *pthread_mutexattr_getpshared()* or *pthread_mutexattr_setpshared()* does not refer to an initialized
55154 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]
55155 error.

55156 **FUTURE DIRECTIONS**

55157 None.

55158 **SEE ALSO**55159 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*, *pthread_mutexattr_destroy()*

55160 XBD <pthread.h>

55161 **CHANGE HISTORY**

55162 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55163 **Issue 6**55164 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions are marked as
55165 part of the Threads and Thread Process-Shared Synchronization options.55166 The **restrict** keyword is added to the *pthread_mutexattr_getpshared()* prototype for alignment
55167 with the ISO/IEC 9899:1999 standard.55168 **Issue 7**55169 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions are marked only
55170 as part of the Thread Process-Shared Synchronization option as the Threads option is now part
55171 of the Base.55172 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
55173 results in undefined behavior.55174 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0281 [972] and XSH/TC2-2008/0282
55175 [757] are applied.

55176 **NAME**

55177 pthread_mutexattr_getrobust, pthread_mutexattr_setrobust ‡ get and set the mutex robust
55178 attribute

55179 **SYNOPSIS**

```
55180 #include <pthread.h>
55181 int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
55182     attr, int *restrict robust);
55183 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
55184     int robust);
```

55185 **DESCRIPTION**

55186 The *pthread_mutexattr_getrobust()* and *pthread_mutexattr_setrobust()* functions, respectively, shall
55187 get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values
55188 for *robust* include:

55189 **PTHREAD_MUTEX_STALLED**

55190 No special actions are taken if the owner of the mutex is terminated while holding the
55191 mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.
55192 This is the default value.

55193 **PTHREAD_MUTEX_ROBUST**

55194 If the process containing the owning thread of a robust mutex terminates while holding the
55195 mutex lock, the next thread that acquires the mutex shall be notified about the termination
55196 by the return value [EOWNERDEAD] from the locking function. If the owning thread of a
55197 robust mutex terminates while holding the mutex lock, the next thread that attempts to
55198 acquire the mutex may be notified about the termination by the return value
55199 [EOWNERDEAD]. The notified thread can then attempt to make the state protected by the
55200 mutex consistent again, and if successful can mark the mutex state as consistent by calling
55201 *pthread_mutex_consistent()*. After a subsequent successful call to *pthread_mutex_unlock()*, the
55202 mutex lock shall be released and can be used normally by other threads. If the mutex is
55203 unlocked without a call to *pthread_mutex_consistent()*, it shall be in a permanently unusable
55204 state and all attempts to lock the mutex shall fail with the error [ENOTRECOVERABLE].
55205 The only permissible operation on such a mutex is *pthread_mutex_destroy()*.

55206 The behavior is undefined if the value specified by the *attr* argument to
55207 *pthread_mutexattr_getrobust()* or *pthread_mutexattr_setrobust()* does not refer to an initialized
55208 mutex attributes object.

55209 **RETURN VALUE**

55210 Upon successful completion, the *pthread_mutexattr_getrobust()* function shall return zero and
55211 store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter.
55212 Otherwise, an error value shall be returned to indicate the error. If successful, the
55213 *pthread_mutexattr_setrobust()* function shall return zero; otherwise, an error number shall be
55214 returned to indicate the error.

55215 **ERRORS**

55216 The *pthread_mutexattr_setrobust()* function shall fail if:

55217 [EINVAL] The value of *robust* is invalid.

55218 These functions shall not return an error code of [EINTR].

55219 EXAMPLES

55220 None.

55221 APPLICATION USAGE

55222 The actions required to make the state protected by the mutex consistent again are solely
55223 dependent on the application. If it is not possible to make the state of a mutex consistent, robust
55224 mutexes can be used to notify this situation by calling *pthread_mutex_unlock()* without a prior
55225 call to *pthread_mutex_consistent()*.

55226 If the state is declared inconsistent by calling *pthread_mutex_unlock()* without a prior call to
55227 *pthread_mutex_consistent()*, a possible approach could be to destroy the mutex and then
55228 reinitialize it. However, it should be noted that this is possible only in certain situations where
55229 the state protected by the mutex has to be reinitialized and coordination achieved with other
55230 threads blocked on the mutex, because otherwise a call to a locking function with a reference to a
55231 mutex object invalidated by a call to *pthread_mutex_destroy()* results in undefined behavior.

55232 RATIONALE

55233 If an implementation detects that the value specified by the *attr* argument to
55234 *pthread_mutexattr_getrobust()* or *pthread_mutexattr_setrobust()* does not refer to an initialized
55235 mutex attributes object, it is recommended that the function should fail and report an [EINVAL]
55236 error.

55237 FUTURE DIRECTIONS

55238 None.

55239 SEE ALSO

55240 [*pthread_mutex_consistent\(\)*](#), [*pthread_mutex_destroy\(\)*](#), [*pthread_mutex_lock\(\)*](#)

55241 XBD <[*pthread.h*](#)>

55242 CHANGE HISTORY

55243 First released in Issue 7.

55244 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0283 [748] is applied.

55245 **NAME**

55246 pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

55247 **SYNOPSIS**

```
55248 #include <pthread.h>
55249 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
55250 int *restrict type);
55251 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

55252 **DESCRIPTION**

55253 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions, respectively, shall get
 55254 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The
 55255 default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.

55256 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types
 55257 include:

```
55258 PTHREAD_MUTEX_NORMAL
55259 PTHREAD_MUTEX_ERRORCHECK
55260 PTHREAD_MUTEX_RECURSIVE
55261 PTHREAD_MUTEX_DEFAULT
```

55262 The mutex type affects the behavior of calls which lock and unlock the mutex. See
 55263 *pthread_mutex_lock()* for details. An implementation may map PTHREAD_MUTEX_DEFAULT
 55264 to one of the other mutex types.

55265 The behavior is undefined if the value specified by the *attr* argument to
 55266 *pthread_mutexattr_gettype()* or *pthread_mutexattr_settype()* does not refer to an initialized mutex
 55267 attributes object.

55268 **RETURN VALUE**

55269 Upon successful completion, the *pthread_mutexattr_gettype()* function shall return zero and store
 55270 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,
 55271 an error shall be returned to indicate the error.

55272 If successful, the *pthread_mutexattr_settype()* function shall return zero; otherwise, an error
 55273 number shall be returned to indicate the error.

55274 **ERRORS**

55275 The *pthread_mutexattr_settype()* function shall fail if:

55276 [EINVAL] The value *type* is invalid.

55277 These functions shall not return an error code of [EINTR].

55278 **EXAMPLES**

55279 None.

55280 **APPLICATION USAGE**

55281 It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with
 55282 condition variables because the implicit unlock performed for a *pthread_cond_timedwait()* or
 55283 *pthread_cond_wait()* may not actually release the mutex (if it had been locked multiple times). If
 55284 this happens, no other thread can satisfy the condition of the predicate.

55285 **RATIONALE**

55286 If an implementation detects that the value specified by the *attr* argument to
 55287 *pthread_mutexattr_gettype()* or *pthread_mutexattr_settype()* does not refer to an initialized mutex
 55288 attributes object, it is recommended that the function should fail and report an [EINVAL] error.

55289 **FUTURE DIRECTIONS**

55290 None.

55291 **SEE ALSO**55292 *pthread_cond_timedwait()*, *pthread_mutex_lock()*

55293 XBD <pthread.h>

55294 **CHANGE HISTORY**

55295 First released in Issue 5.

55296 **Issue 6**55297 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for
55298 *pthread_mutexattr_gettype()* is updated so that the first argument is of type **const**
55299 **pthread_mutexattr_t***.55300 The **restrict** keyword is added to the *pthread_mutexattr_gettype()* prototype for alignment with
55301 the ISO/IEC 9899:1999 standard.55302 **Issue 7**55303 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions are moved from the
55304 XSI option to the Base.55305 The [EINVAL] error for an uninitialized mutex attributes object is removed; this condition
55306 results in undefined behavior.

55307 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0464 [121] is applied.

55308 **NAME**

55309 pthread_mutexattr_init — initialize the mutex attributes object

55310 **SYNOPSIS**

55311 #include <pthread.h>

55312 int pthread_mutexattr_init(pthread_mutexattr_t *attr);

55313 **DESCRIPTION**

55314 Refer to *pthread_mutexattr_destroy()*.

55315 **NAME**

55316 pthread_mutexattr_setprioceiling ‡'set the prioceiling attribute of the mutex attributes object
55317 (**REALTIME THREADS**)

55318 **SYNOPSIS**

```
55319 RPP|TPP #include <pthread.h>  
55320 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
55321 int prioceiling);
```

55322 **DESCRIPTION**

55323 Refer to [pthread_mutexattr_getprioceiling\(\)](#).

55324 **NAME**

55325 pthread_mutexattr_setprotocol ¶ set the protocol attribute of the mutex attributes object
55326 (**REALTIME THREADS**)

55327 **SYNOPSIS**

```
55328 MC1 #include <pthread.h>  
55329 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,  
55330 int protocol);
```

55331 **DESCRIPTION**

55332 Refer to [pthread_mutexattr_getprotocol\(\)](#).

55333 **NAME**

55334 pthread_mutexattr_setpshared — set the process-shared attribute

55335 **SYNOPSIS**

```
55336 TSH #include <pthread.h>
55337      int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
55338      int pshared);
```

55339 **DESCRIPTION**55340 Refer to [pthread_mutexattr_getpshared\(\)](#).

55341 **NAME**

55342 pthread_mutexattr_setrobust — get and set the mutex robust attribute

55343 **SYNOPSIS**

55344 #include <pthread.h>

55345 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
55346 int robust);

55347 **DESCRIPTION**

55348 Refer to [pthread_mutexattr_getrobust\(\)](#).

55349 **NAME**

55350 pthread_mutexattr_settype — set the mutex type attribute

55351 **SYNOPSIS**

55352 #include <pthread.h>

55353 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);

55354 **DESCRIPTION**55355 Refer to *pthread_mutexattr_gettype()*.

55356 **NAME**

55357 pthread_once — dynamic package initialization

55358 **SYNOPSIS**

```
55359 #include <pthread.h>
55360 int pthread_once(pthread_once_t *once_control,
55361                 void (*init_routine)(void));
55362 pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

55363 **DESCRIPTION**

55364 The first call to *pthread_once()* by any thread in a process, with a given *once_control*, shall call the
 55365 *init_routine* with no arguments. Subsequent calls of *pthread_once()* with the same *once_control*
 55366 shall not call the *init_routine*. On return from *pthread_once()*, *init_routine* shall have completed.
 55367 The *once_control* parameter shall determine whether the associated initialization routine has been
 55368 called.

55369 The *pthread_once()* function is not a cancellation point. However, if *init_routine* is a cancellation
 55370 point and is canceled, the effect on *once_control* shall be as if *pthread_once()* was never called.

55371 If the call to *init_routine* is terminated by a call to *longjmp()*, *_longjmp()*, or *siglongjmp()*, the
 55372 behavior is undefined.

55373 The constant PTHREAD_ONCE_INIT is defined in the **<pthread.h>** header.

55374 The behavior of *pthread_once()* is undefined if *once_control* has automatic storage duration or is
 55375 not initialized by PTHREAD_ONCE_INIT.

55376 **RETURN VALUE**

55377 Upon successful completion, *pthread_once()* shall return zero; otherwise, an error number shall
 55378 be returned to indicate the error.

55379 **ERRORS**

55380 The *pthread_once()* function shall not return an error code of [EINTR].

55381 **EXAMPLES**

55382 None.

55383 **APPLICATION USAGE**

55384 If *init_routine* recursively calls *pthread_once()* with the same *once_control*, the recursive call will
 55385 not call the specified *init_routine*, and thus the specified *init_routine* will not complete, and thus
 55386 the recursive call to *pthread_once()* will not return. Use of *longjmp()*, *_longjmp()*, or *siglongjmp()*
 55387 within an *init_routine* to jump to a point outside of *init_routine* prevents *init_routine* from
 55388 returning.

55389 **RATIONALE**

55390 Some C libraries are designed for dynamic initialization. That is, the global initialization for the
 55391 library is performed when the first procedure in the library is called. In a single-threaded
 55392 program, this is normally implemented using a static variable whose value is checked on entry
 55393 to a routine, as follows:

```
55394 static int random_is_initialized = 0;
55395 extern void initialize_random(void);
55396
55397 int random_function()
55398 {
55398     if (random_is_initialized == 0) {
55399         initialize_random();
55400         random_is_initialized = 1;
```

```

55401     }
55402     ... /* Operations performed after initialization. */
55403 }

```

55404 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,
 55405 library initialization has to be accomplished by an explicit call to a library-exported initialization
 55406 function prior to any use of the library.

55407 For dynamic library initialization in a multi-threaded process, if an initialization flag is used the
 55408 flag needs to be protected against modification by multiple threads simultaneously calling into
 55409 the library. This can be done by using a mutex (initialized by assigning
 55410 PTHREAD_MUTEX_INITIALIZER). However, the better solution is to use *pthread_once()* which
 55411 is designed for exactly this purpose, as follows:

```

55412 #include <pthread.h>
55413 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
55414 extern void initialize_random(void);

55415 int random_function()
55416 {
55417     (void) pthread_once(&random_is_initialized, initialize_random);
55418     ... /* Operations performed after initialization. */
55419 }

```

55420 If an implementation detects that the value specified by the *once_control* argument to
 55421 *pthread_once()* does not refer to a **pthread_once_t** object initialized by PTHREAD_ONCE_INIT,
 55422 it is recommended that the function should fail and report an [EINVAL] error.

55423 FUTURE DIRECTIONS

55424 None.

55425 SEE ALSO

55426 XBD [<pthread.h>](#)

55427 CHANGE HISTORY

55428 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

55429 Issue 6

55430 The *pthread_once()* function is marked as part of the Threads option.

55431 The [EINVAL] error is added as a “may fail” case for if either argument is invalid.

55432 Issue 7

55433 The *pthread_once()* function is moved from the Threads option to the Base.

55434 The [EINVAL] error for an uninitialized **pthread_once_t** object is removed; this condition results
 55435 in undefined behavior.

55436 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0284 [863], XSH/TC2-2008/0285 [874],
 55437 XSH/TC2-2008/0286 [874], and XSH/TC2-2008/0287 [747] are applied.

55438 **NAME**

55439 pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

55440 **SYNOPSIS**

```
55441 #include <pthread.h>
55442 int pthread_rwlock_destroy(pthread_rwlock_t *rwlck);
55443 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlck,
55444     const pthread_rwlockattr_t *restrict attr);
55445 pthread_rwlock_t rwlck = PTHREAD_RWLOCK_INITIALIZER;
```

55446 **DESCRIPTION**

55447 The *pthread_rwlock_destroy()* function shall destroy the read-write lock object referenced by
 55448 *rwlck* and release any resources used by the lock. The effect of subsequent use of the lock is
 55449 undefined until the lock is reinitialized by another call to *pthread_rwlock_init()*. An
 55450 implementation may cause *pthread_rwlock_destroy()* to set the object referenced by *rwlck* to an
 55451 invalid value. Results are undefined if *pthread_rwlock_destroy()* is called when any thread holds
 55452 *rwlck*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

55453 The *pthread_rwlock_init()* function shall allocate any resources required to use the read-write lock
 55454 referenced by *rwlck* and initializes the lock to an unlocked state with attributes referenced by
 55455 *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as
 55456 passing the address of a default read-write lock attributes object. Once initialized, the lock can be
 55457 used any number of times without being reinitialized. Results are undefined if
 55458 *pthread_rwlock_init()* is called specifying an already initialized read-write lock. Results are
 55459 undefined if a read-write lock is used without first being initialized.

55460 If the *pthread_rwlock_init()* function fails, *rwlck* shall not be initialized and the contents of *rwlck*
 55461 are undefined.

55462 See [Section 2.9.9](#) (on page 523) for further requirements.

55463 In cases where default read-write lock attributes are appropriate, the macro
 55464 PTHREAD_RWLOCK_INITIALIZER can be used to initialize read-write locks. The effect shall
 55465 be equivalent to dynamic initialization by a call to *pthread_rwlock_init()* with the *attr* parameter
 55466 specified as NULL, except that no error checks are performed.

55467 The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlock_init()*
 55468 does not refer to an initialized read-write lock attributes object.

55469 **RETURN VALUE**

55470 If successful, the *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions shall return zero;
 55471 otherwise, an error number shall be returned to indicate the error.

55472 **ERRORS**

55473 The *pthread_rwlock_init()* function shall fail if:

55474 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 55475 another read-write lock.

55476 [ENOMEM] Insufficient memory exists to initialize the read-write lock.

55477 [EPERM] The caller does not have the privilege to perform the operation.

55478 These functions shall not return an error code of [EINTR].

55479 **EXAMPLES**

55480 None.

55481 **APPLICATION USAGE**55482 Applications using these and related read-write lock functions may be subject to priority
55483 inversion, as discussed in XBD [Section 3.291](#) (on page 80).55484 **RATIONALE**55485 If an implementation detects that the value specified by the *rwlock* argument to
55486 *pthread_rwlock_destroy()* does not refer to an initialized read-write lock object, it is recommended
55487 that the function should fail and report an [EINVAL] error.55488 If an implementation detects that the value specified by the *attr* argument to
55489 *pthread_rwlock_init()* does not refer to an initialized read-write lock attributes object, it is
55490 recommended that the function should fail and report an [EINVAL] error.55491 If an implementation detects that the value specified by the *rwlock* argument to
55492 *pthread_rwlock_destroy()* or *pthread_rwlock_init()* refers to a locked read-write lock object, or
55493 detects that the value specified by the *rwlock* argument to *pthread_rwlock_init()* refers to an
55494 already initialized read-write lock object, it is recommended that the function should fail and
55495 report an [EBUSY] error.55496 **FUTURE DIRECTIONS**

55497 None.

55498 **SEE ALSO**55499 [pthread_rwlock_rdlock\(\)](#), [pthread_rwlock_timedrdlock\(\)](#), [pthread_rwlock_timedwrlock\(\)](#),
55500 [pthread_rwlock_trywrlock\(\)](#), [pthread_rwlock_unlock\(\)](#)55501 XBD [Section 3.291](#) (on page 80), [<pthread.h>](#)55502 **CHANGE HISTORY**

55503 First released in Issue 5.

55504 **Issue 6**

55505 The following changes are made for alignment with IEEE Std 1003.1j-2000:

55506 The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
55507 now part of the Threads option (previously it was part of the Read-Write Locks option in
55508 IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also
55509 deleted from the SYNOPSIS.

55510 The DESCRIPTION is updated as follows:

55511 ‡ ~~Explicitly~~ notes allocation of resources upon initialization of a read-write lock
55512 object.55513 ‡ ~~A~~ paragraph is added specifying that copies of read-write lock objects may not be
55514 used.55515 An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init()*, indicating
55516 that the *rwlock* value is invalid.

55517 The SEE ALSO section is updated.

55518 The **restrict** keyword is added to the *pthread_rwlock_init()* prototype for alignment with the
55519 ISO/IEC 9899:1999 standard.55520 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION
55521 USAGE relating to priority inversion.

55522 **Issue 7**

55523 Austin Group Interpretation 1003.1-2001 #048 is applied, adding the
55524 PTHREAD_RWLOCK_INITIALIZER macro.

55525 The *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions are moved from the Threads
55526 option to the Base.

55527 The [EINVAL] error for an uninitialized read-write lock object or read-write lock attributes
55528 object is removed; this condition results in undefined behavior.

55529 The [EBUSY] error for a locked read-write lock object or an already initialized read-write lock
55530 object is removed; this condition results in undefined behavior.

55531 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0465 [70] is applied.

55532 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0288 [972] and XSH/TC2-2008/0289
55533 [758] are applied.

55534 **NAME**

55535 pthread_rwlock_rdlock, pthread_rwlock_tryrdlock — lock a read-write lock object for reading

55536 **SYNOPSIS**

55537 #include <pthread.h>

55538 int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);

55539 int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);

55540 **DESCRIPTION**55541 The *pthread_rwlock_rdlock()* function shall apply a read lock to the read-write lock referenced by
55542 *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are
55543 no writers blocked on the lock.55544 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
55545 executing with the scheduling policies SCHED_FIFO or SCHED_RR, the calling thread shall not
55546 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on
55547 the lock; otherwise, the calling thread shall acquire the lock.55548 TPS TSP If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
55549 executing with the SCHED_SPORADIC scheduling policy, the calling thread shall not acquire
55550 the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock;
55551 otherwise, the calling thread shall acquire the lock.55552 If the Thread Execution Scheduling option is not supported, it is implementation-defined
55553 whether the calling thread acquires the lock when a writer does not hold the lock and there are
55554 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the
55555 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the
55556 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.55557 A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the
55558 *pthread_rwlock_rdlock()* function *n* times). If so, the application shall ensure that the thread
55559 performs matching unlocks (that is, it calls the *pthread_rwlock_unlock()* function *n* times).55560 The maximum number of simultaneous read locks that an implementation guarantees can be
55561 applied to a read-write lock shall be implementation-defined. The *pthread_rwlock_rdlock()*
55562 function may fail if this maximum would be exceeded.55563 The *pthread_rwlock_tryrdlock()* function shall apply a read lock as in the *pthread_rwlock_rdlock()*
55564 function, with the exception that the function shall fail if the equivalent *pthread_rwlock_rdlock()*
55565 call would have blocked the calling thread. In no case shall the *pthread_rwlock_tryrdlock()*
55566 function ever block; it always either acquires the lock or fails and returns immediately.

55567 Results are undefined if any of these functions are called with an uninitialized read-write lock.

55568 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the
55569 signal handler the thread resumes waiting for the read-write lock for reading as if it was not
55570 interrupted.55571 **RETURN VALUE**55572 If successful, the *pthread_rwlock_rdlock()* function shall return zero; otherwise, an error number
55573 shall be returned to indicate the error.55574 The *pthread_rwlock_tryrdlock()* function shall return zero if the lock for reading on the read-write
55575 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
55576 indicate the error.

55577 **ERRORS**55578 The *pthread_rwlock_tryrdlock()* function shall fail if:55579 [EBUSY] The read-write lock could not be acquired for reading because a writer holds
55580 the lock or a writer with the appropriate priority was blocked on it.55581 The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions may fail if:55582 [EAGAIN] The read lock could not be acquired because the maximum number of read
55583 locks for *rwlock* has been exceeded.55584 The *pthread_rwlock_rdlock()* function may fail if:55585 [EDEADLK] A deadlock condition was detected or the current thread already owns the
55586 read-write lock for writing.

55587 These functions shall not return an error code of [EINTR].

55588 **EXAMPLES**

55589 None.

55590 **APPLICATION USAGE**55591 Applications using these functions may be subject to priority inversion, as discussed in XBD
55592 [Section 3.291](#) (on page 80).55593 **RATIONALE**55594 If an implementation detects that the value specified by the *rwlock* argument to
55595 *pthread_rwlock_rdlock()* or *pthread_rwlock_tryrdlock()* does not refer to an initialized read-write
55596 lock object, it is recommended that the function should fail and report an [EINVAL] error.55597 **FUTURE DIRECTIONS**

55598 None.

55599 **SEE ALSO**55600 *pthread_rwlock_destroy()*, *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*,
55601 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*55602 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<pthread.h>](#)55603 **CHANGE HISTORY**

55604 First released in Issue 5.

55605 **Issue 6**

55606 The following changes are made for alignment with IEEE Std 1003.1j-2000:

55607 The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
55608 now part of the Threads option (previously it was part of the Read-Write Locks option in
55609 IEEE Std 1003.1j-2000 and also part of the XSI extension).

55610 The DESCRIPTION is updated as follows:

55611 ‡ **C**onditions under which writers have precedence over readers are specified.55612 ‡ **A**ailure of *pthread_rwlock_tryrdlock()* is clarified.55613 ‡ **P**aragraph on the maximum number of read locks is added.55614 In the ERRORS sections, [EBUSY] is modified to take into account write priority, and
55615 [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock()* error.

- 55616 The SEE ALSO section is updated.
- 55617 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/101 is applied, updating the ERRORS
55618 section so that the [EDEADLK] error includes detection of a deadlock condition.
- 55619 **Issue 7**
- 55620 The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions are moved from the Threads
55621 option to the Base.
- 55622 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
55623 in undefined behavior.

55624 **NAME**

55625 pthread_rwlock_timedrdlock — lock a read-write lock for reading

55626 **SYNOPSIS**

55627 #include <pthread.h>

55628 #include <time.h>

```
55629 int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rlock,
55630                               const struct timespec *restrict abstime);
```

55631 **DESCRIPTION**

55632 The *pthread_rwlock_timedrdlock()* function shall apply a read lock to the read-write lock
 55633 referenced by *rlock* as in the *pthread_rwlock_rdlock()* function. However, if the lock cannot be
 55634 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
 55635 when the specified timeout expires. The timeout shall expire when the absolute time specified
 55636 by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value
 55637 of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already
 55638 been passed at the time of the call.

55639 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 55640 be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the
 55641 <**time.h**> header. Under no circumstances shall the function fail with a timeout if the lock can be
 55642 acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can
 55643 be immediately acquired.

55644 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
 55645 write lock via a call to *pthread_rwlock_timedrdlock()*, upon return from the signal handler the
 55646 thread shall resume waiting for the lock as if it was not interrupted.

55647 The calling thread may deadlock if at the time the call is made it holds a write lock on *rlock*.
 55648 The results are undefined if this function is called with an uninitialized read-write lock.

55649 **RETURN VALUE**

55650 The *pthread_rwlock_timedrdlock()* function shall return zero if the lock for reading on the read-
 55651 write lock object referenced by *rlock* is acquired. Otherwise, an error number shall be returned
 55652 to indicate the error.

55653 **ERRORS**55654 The *pthread_rwlock_timedrdlock()* function shall fail if:

55655 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

55656 The *pthread_rwlock_timedrdlock()* function may fail if:55657 [EAGAIN] The read lock could not be acquired because the maximum number of read
55658 locks for lock would be exceeded.55659 [EDEADLK] A deadlock condition was detected or the calling thread already holds a write
55660 lock on *rlock*.55661 [EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000
55662 million.

55663 This function shall not return an error code of [EINTR].

55664 **EXAMPLES**

55665 None.

55666 **APPLICATION USAGE**55667 Applications using this function may be subject to priority inversion, as discussed in XBD
55668 [Section 3.291](#) (on page 80).55669 **RATIONALE**55670 If an implementation detects that the value specified by the *rwlock* argument to
55671 *pthread_rwlock_timedrdlock()* does not refer to an initialized read-write lock object, it is
55672 recommended that the function should fail and report an [EINVAL] error.55673 **FUTURE DIRECTIONS**

55674 None.

55675 **SEE ALSO**55676 [pthread_rwlock_destroy\(\)](#), [pthread_rwlock_rdlock\(\)](#), [pthread_rwlock_timedwrlock\(\)](#),
55677 [pthread_rwlock_trywrlock\(\)](#), [pthread_rwlock_unlock\(\)](#)55678 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<pthread.h>](#), [<time.h>](#)55679 **CHANGE HISTORY**

55680 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55681 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/102 is applied, updating the ERRORS
55682 section so that the [EDEADLK] error includes detection of a deadlock condition.55683 **Issue 7**55684 The *pthread_rwlock_timedrdlock()* function is moved from the Timeouts option to the Base.55685 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
55686 in undefined behavior.

55687 **NAME**

55688 pthread_rwlock_timedwrlock — lock a read-write lock for writing

55689 **SYNOPSIS**

55690 #include <pthread.h>

55691 #include <time.h>

```
55692 int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rlock,
55693                               const struct timespec *restrict abstime);
```

55694 **DESCRIPTION**

55695 The *pthread_rwlock_timedwrlock()* function shall apply a write lock to the read-write lock
 55696 referenced by *rlock* as in the *pthread_rwlock_wrlock()* function. However, if the lock cannot be
 55697 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
 55698 when the specified timeout expires. The timeout shall expire when the absolute time specified
 55699 by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value
 55700 of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already
 55701 been passed at the time of the call.

55702 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 55703 be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the
 55704 <**time.h**> header. Under no circumstances shall the function fail with a timeout if the lock can be
 55705 acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can
 55706 be immediately acquired.

55707 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
 55708 write lock via a call to *pthread_rwlock_timedwrlock()*, upon return from the signal handler the
 55709 thread shall resume waiting for the lock as if it was not interrupted.

55710 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The
 55711 results are undefined if this function is called with an uninitialized read-write lock.

55712 **RETURN VALUE**

55713 The *pthread_rwlock_timedwrlock()* function shall return zero if the lock for writing on the read-
 55714 write lock object referenced by *rlock* is acquired. Otherwise, an error number shall be returned
 55715 to indicate the error.

55716 **ERRORS**

55717 The *pthread_rwlock_timedwrlock()* function shall fail if:

55718 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

55719 The *pthread_rwlock_timedwrlock()* function may fail if:

55720 [EDEADLK] A deadlock condition was detected or the calling thread already holds the
 55721 *rlock*.

55722 [EINVAL] The *abstime* nanosecond value is less than zero or greater than or equal to 1 000
 55723 million.

55724 This function shall not return an error code of [EINTR].

55725 **EXAMPLES**

55726 None.

55727 **APPLICATION USAGE**

55728 Applications using this function may be subject to priority inversion, as discussed in XBD
55729 [Section 3.291](#) (on page 80).

55730 **RATIONALE**

55731 If an implementation detects that the value specified by the *rwlock* argument to
55732 *pthread_rwlock_timedwrlock()* does not refer to an initialized read-write lock object, it is
55733 recommended that the function should fail and report an [EINVAL] error.

55734 **FUTURE DIRECTIONS**

55735 None.

55736 **SEE ALSO**

55737 [pthread_rwlock_destroy\(\)](#), [pthread_rwlock_rdlock\(\)](#), [pthread_rwlock_timedrdlock\(\)](#),
55738 [pthread_rwlock_trywrlock\(\)](#), [pthread_rwlock_unlock\(\)](#)

55739 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<pthread.h>](#), [<time.h>](#)55740 **CHANGE HISTORY**

55741 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

55742 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/103 is applied, updating the ERRORS
55743 section so that the [EDEADLK] error includes detection of a deadlock condition.

55744 **Issue 7**55745 The *pthread_rwlock_timedwrlock()* function is moved from the Timeouts option to the Base.

55746 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
55747 in undefined behavior.

55748 **NAME**

55749 pthread_rwlock_tryrdlock — lock a read-write lock object for reading

55750 **SYNOPSIS**

55751 #include <pthread.h>

55752 int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);

55753 **DESCRIPTION**

55754 Refer to *pthread_rwlock_rdlock()*.

55755 **NAME**

55756 pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

55757 **SYNOPSIS**

55758 #include <pthread.h>

55759 int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);

55760 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

55761 **DESCRIPTION**55762 The *pthread_rwlock_trywrlock()* function shall apply a write lock like the *pthread_rwlock_wrlock()*
55763 function, with the exception that the function shall fail if any thread currently holds *rwlock* (for
55764 reading or writing).55765 The *pthread_rwlock_wrlock()* function shall apply a write lock to the read-write lock referenced by
55766 *rwlock*. The calling thread shall acquire the write lock if no thread (reader or writer) holds the
55767 read-write lock *rwlock*. Otherwise, if another thread holds the read-write lock *rwlock*, the calling
55768 thread shall block until it can acquire the lock. If a deadlock condition occurs or the calling
55769 thread already owns the read-write lock for writing or reading, the call shall either deadlock or
55770 return [EDEADLK].

55771 Results are undefined if any of these functions are called with an uninitialized read-write lock.

55772 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the
55773 signal handler the thread resumes waiting for the read-write lock for writing as if it was not
55774 interrupted.55775 **RETURN VALUE**55776 The *pthread_rwlock_trywrlock()* function shall return zero if the lock for writing on the read-write
55777 lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to
55778 indicate the error.55779 If successful, the *pthread_rwlock_wrlock()* function shall return zero; otherwise, an error number
55780 shall be returned to indicate the error.55781 **ERRORS**55782 The *pthread_rwlock_trywrlock()* function shall fail if:55783 [EBUSY] The read-write lock could not be acquired for writing because it was already
55784 locked for reading or writing.55785 The *pthread_rwlock_wrlock()* function may fail if:55786 [EDEADLK] A deadlock condition was detected or the current thread already owns the
55787 read-write lock for writing or reading.

55788 These functions shall not return an error code of [EINTR].

55789 **EXAMPLES**

55790 None.

55791 **APPLICATION USAGE**55792 Applications using these functions may be subject to priority inversion, as discussed in XBD
55793 [Section 3.291](#) (on page 80).55794 **RATIONALE**55795 If an implementation detects that the value specified by the *rwlock* argument to
55796 *pthread_rwlock_trywrlock()* or *pthread_rwlock_wrlock()* does not refer to an initialized read-write
55797 lock object, it is recommended that the function should fail and report an [EINVAL] error.

55798 **FUTURE DIRECTIONS**

55799 None.

55800 **SEE ALSO**55801 *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
55802 *pthread_rwlock_timedwrlock()*, *pthread_rwlock_unlock()*55803 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<pthread.h>](#)55804 **CHANGE HISTORY**

55805 First released in Issue 5.

55806 **Issue 6**

55807 The following changes are made for alignment with IEEE Std 1003.1j-2000:

55808 The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
55809 now part of the Threads option (previously it was part of the Read-Write Locks option in
55810 IEEE Std 1003.1j-2000 and also part of the XSI extension).55811 The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock()* error.

55812 The SEE ALSO section is updated.

55813 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/104 is applied, updating the ERRORS
55814 section so that the [EDEADLK] error includes detection of a deadlock condition.55815 **Issue 7**55816 The *pthread_rwlock_trywrlock()* and *pthread_rwlock_wrlock()* functions are moved from the
55817 Threads option to the Base.55818 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
55819 in undefined behavior.55820 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0290 [720] and XSH/TC2-2008/0291
55821 [722] are applied.

55822 **NAME**

55823 pthread_rwlock_unlock — unlock a read-write lock object

55824 **SYNOPSIS**

55825 #include <pthread.h>

55826 int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);

55827 **DESCRIPTION**

55828 The *pthread_rwlock_unlock()* function shall release a lock held on the read-write lock object
55829 referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the
55830 calling thread.

55831 If this function is called to release a read lock from the read-write lock object and there are other
55832 read locks currently held on this read-write lock object, the read-write lock object remains in the
55833 read locked state. If this function releases the last read lock for this read-write lock object, the
55834 read-write lock object shall be put in the unlocked state with no owners.

55835 If this function is called to release a write lock for this read-write lock object, the read-write lock
55836 object shall be put in the unlocked state.

55837 If there are threads blocked on the lock when it becomes available, the scheduling policy shall
55838 TPS determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is
55839 supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or
55840 SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when
55841 the lock becomes available. For equal priority threads, write locks shall take precedence over
55842 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-
55843 defined whether write locks take precedence over read locks.

55844 Results are undefined if this function is called with an uninitialized read-write lock.

55845 **RETURN VALUE**

55846 If successful, the *pthread_rwlock_unlock()* function shall return zero; otherwise, an error number
55847 shall be returned to indicate the error.

55848 **ERRORS**

55849 The *pthread_rwlock_unlock()* function shall not return an error code of [EINTR].

55850 **EXAMPLES**

55851 None.

55852 **APPLICATION USAGE**

55853 None.

55854 **RATIONALE**

55855 If an implementation detects that the value specified by the *rwlock* argument to
55856 *pthread_rwlock_unlock()* does not refer to an initialized read-write lock object, it is recommended
55857 that the function should fail and report an [EINVAL] error.

55858 If an implementation detects that the value specified by the *rwlock* argument to
55859 *pthread_rwlock_unlock()* refers to a read-write lock object for which the current thread does not
55860 hold a lock, it is recommended that the function should fail and report an [EPERM] error.

55861 **FUTURE DIRECTIONS**

55862 None.

55863 **SEE ALSO**

55864 *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
55865 *pthread_rwlock_timedwrlock()*, *pthread_rwlock_trywrlock()*

55866 XBD Section 4.12 (on page 111), <pthread.h>

55867 **CHANGE HISTORY**

55868 First released in Issue 5.

55869 **Issue 6**

55870 The following changes are made for alignment with IEEE Std 1003.1j-2000:

55871 The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
55872 now part of the Threads option (previously it was part of the Read-Write Locks option in
55873 IEEE Std 1003.1j-2000 and also part of the XSI extension).

55874 The DESCRIPTION is updated as follows:

55875 ‡ The conditions under which writers have precedence over readers are specified.

55876 ‡ The concept of read-write lock owner is deleted.

55877 The SEE ALSO section is updated.

55878 **Issue 7**

55879 SD5-XSH-ERN-183 is applied.

55880 The *pthread_rwlock_unlock()* function is moved from the Threads option to the Base.

55881 The [EINVAL] error for an uninitialized read-write lock object is removed; this condition results
55882 in undefined behavior.

55883 The [EPERM] error for a read-write lock object for which the current thread does not hold a lock
55884 is removed; this condition results in undefined behavior.

55885 **NAME**

55886 pthread_rwlock_wrlock — lock a read-write lock object for writing

55887 **SYNOPSIS**

55888 #include <pthread.h>

55889 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

55890 **DESCRIPTION**55891 Refer to *pthread_rwlock_trywrlock()*.

55892 **NAME**

55893 pthread_rwlockattr_destroy, pthread_rwlockattr_init ‡' destroy and initialize the read-write
55894 lock attributes object

55895 **SYNOPSIS**

```
55896 #include <pthread.h>
55897 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
55898 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

55899 **DESCRIPTION**

55900 The *pthread_rwlockattr_destroy()* function shall destroy a read-write lock attributes object. A
55901 destroyed *attr* attributes object can be reinitialized using *pthread_rwlockattr_init()*; the results of
55902 otherwise referencing the object after it has been destroyed are undefined. An implementation
55903 may cause *pthread_rwlockattr_destroy()* to set the object referenced by *attr* to an invalid value.

55904 The *pthread_rwlockattr_init()* function shall initialize a read-write lock attributes object *attr* with
55905 the default value for all of the attributes defined by the implementation.

55906 Results are undefined if *pthread_rwlockattr_init()* is called specifying an already initialized *attr*
55907 attributes object.

55908 After a read-write lock attributes object has been used to initialize one or more read-write locks,
55909 any function affecting the attributes object (including destruction) shall not affect any previously
55910 initialized read-write locks.

55911 The behavior is undefined if the value specified by the *attr* argument to
55912 *pthread_rwlockattr_destroy()* does not refer to an initialized read-write lock attributes object.

55913 **RETURN VALUE**

55914 If successful, the *pthread_rwlockattr_destroy()* and *pthread_rwlockattr_init()* functions shall return
55915 zero; otherwise, an error number shall be returned to indicate the error.

55916 **ERRORS**

55917 The *pthread_rwlockattr_init()* function shall fail if:

55918 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

55919 These functions shall not return an error code of [EINTR].

55920 **EXAMPLES**

55921 None.

55922 **APPLICATION USAGE**

55923 None.

55924 **RATIONALE**

55925 If an implementation detects that the value specified by the *attr* argument to
55926 *pthread_rwlockattr_destroy()* does not refer to an initialized read-write lock attributes object, it is
55927 recommended that the function should fail and report an [EINVAL] error.

55928 **FUTURE DIRECTIONS**

55929 None.

55930 **SEE ALSO**

55931 *pthread_rwlock_destroy()*, *pthread_rwlockattr_getpshared()*

55932 XBD <pthread.h>

55933 **CHANGE HISTORY**

55934 First released in Issue 5.

55935 **Issue 6**

55936 The following changes are made for alignment with IEEE Std 1003.1j-2000:

55937 The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is
55938 now part of the Threads option (previously it was part of the Read-Write Locks option in
55939 IEEE Std 1003.1j-2000 and also part of the XSI extension).

55940 The SEE ALSO section is updated.

55941 **Issue 7**55942 The *pthread_rwlockattr_destroy()* and *pthread_rwlockattr_init()* functions are moved from the
55943 Threads option to the Base.55944 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this
55945 condition results in undefined behavior.

55946 **NAME**

55947 pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared ‡ get and set the process-
 55948 shared attribute of the read-write lock attributes object

55949 **SYNOPSIS**

```
55950 TSH #include <pthread.h>
55951
55952 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
55953 *restrict attr, int *restrict pshared);
55954 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
55955 int pshared);
```

55955 **DESCRIPTION**

55956 The *pthread_rwlockattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 55957 from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared()*
 55958 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

55959 The *process-shared* attribute shall be set to `PTHREAD_PROCESS_SHARED` to permit a read-write
 55960 lock to be operated upon by any thread that has access to the memory where the read-write lock
 55961 is allocated, even if the read-write lock is allocated in memory that is shared by multiple
 55962 processes. See [Section 2.9.9](#) (on page 523) for further requirements. The default value of the
 55963 *process-shared* attribute shall be `PTHREAD_PROCESS_PRIVATE`.

55964 Additional attributes, their default values, and the names of the associated functions to get and
 55965 set those attribute values are implementation-defined.

55966 The behavior is undefined if the value specified by the *attr* argument to
 55967 *pthread_rwlockattr_getpshared()* or *pthread_rwlockattr_setpshared()* does not refer to an initialized
 55968 read-write lock attributes object.

55969 **RETURN VALUE**

55970 Upon successful completion, the *pthread_rwlockattr_getpshared()* function shall return zero and
 55971 store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared*
 55972 parameter. Otherwise, an error number shall be returned to indicate the error.

55973 If successful, the *pthread_rwlockattr_setpshared()* function shall return zero; otherwise, an error
 55974 number shall be returned to indicate the error.

55975 **ERRORS**

55976 The *pthread_rwlockattr_setpshared()* function may fail if:

55977 [EINVAL] The new value specified for the attribute is outside the range of legal values
 55978 for that attribute.

55979 These functions shall not return an error code of [EINTR].

55980 **EXAMPLES**

55981 None.

55982 **APPLICATION USAGE**

55983 None.

55984 **RATIONALE**

55985 None.

55986 **FUTURE DIRECTIONS**

55987 None.

55988 **SEE ALSO**55989 *pthread_rwlock_destroy()*, *pthread_rwlockattr_destroy()*

55990 XBD <pthread.h>

55991 **CHANGE HISTORY**

55992 First released in Issue 5.

55993 **Issue 6**

55994 The following changes are made for alignment with IEEE Std 1003.1j-2000:

55995 The margin code in the SYNOPSIS is changed to THR TSH to indicate that the
55996 functionality is now part of the Threads option (previously it was part of the Read-Write
55997 Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).

55998 The DESCRIPTION notes that additional attributes are implementation-defined.

55999 The SEE ALSO section is updated.

56000 The **restrict** keyword is added to the *pthread_rwlockattr_getpshared()* prototype for alignment
56001 with the ISO/IEC 9899:1999 standard.

56002 **Issue 7**

56003 The *pthread_rwlockattr_getpshared()* and *pthread_rwlockattr_setpshared()* functions are marked
56004 only as part of the Thread Process-Shared Synchronization option as the Threads option is now
56005 part of the Base.

56006 The [EINVAL] error for an uninitialized read-write lock attributes object is removed; this
56007 condition results in undefined behavior.

56008 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0292 [972] and XSH/TC2-2008/0293
56009 [757] are applied.

56010 **NAME**

56011 pthread_rwlockattr_init — initialize the read-write lock attributes object

56012 **SYNOPSIS**

56013 #include <pthread.h>

56014 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

56015 **DESCRIPTION**

56016 Refer to *pthread_rwlockattr_destroy()*.

56017 **NAME**

56018 pthread_rwlockattr_setpshared ¶ set the process-shared attribute of the read-write lock
56019 attributes object

56020 **SYNOPSIS**

```
56021 TSH #include <pthread.h>  
56022 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,  
56023 int pshared);
```

56024 **DESCRIPTION**

56025 Refer to [pthread_rwlockattr_getpshared\(\)](#).

56026 **NAME**

56027 pthread_self — get the calling thread ID

56028 **SYNOPSIS**

56029 #include <pthread.h>

56030 pthread_t pthread_self(void);

56031 **DESCRIPTION**

56032 The *pthread_self()* function shall return the thread ID of the calling thread.

56033 **RETURN VALUE**

56034 The *pthread_self()* function shall always be successful and no return value is reserved to indicate
56035 an error.

56036 **ERRORS**

56037 No errors are defined.

56038 **EXAMPLES**

56039 None.

56040 **APPLICATION USAGE**

56041 None.

56042 **RATIONALE**

56043 The *pthread_self()* function provides a capability similar to the *getpid()* function for processes
56044 and the rationale is the same: the creation call does not provide the thread ID to the created
56045 thread.

56046 **FUTURE DIRECTIONS**

56047 None.

56048 **SEE ALSO**

56049 *pthread_create()*, *pthread_equal()*

56050 XBD <pthread.h>

56051 **CHANGE HISTORY**

56052 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

56053 **Issue 6**

56054 The *pthread_self()* function is marked as part of the Threads option.

56055 **Issue 7**

56056 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section.

56057 The *pthread_self()* function is moved from the Threads option to the Base.

56058 **NAME**

56059 pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

56060 **SYNOPSIS**

```
56061 #include <pthread.h>
56062 int pthread_setcancelstate(int state, int *oldstate);
56063 int pthread_setcanceltype(int type, int *oldtype);
56064 void pthread_testcancel(void);
```

56065 **DESCRIPTION**

56066 The *pthread_setcancelstate()* function shall atomically both set the calling thread's cancelability state to the indicated *state* and return the previous cancelability state at the location referenced by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DISABLE.

56070 The *pthread_setcanceltype()* function shall atomically both set the calling thread's cancelability type to the indicated *type* and return the previous cancelability type at the location referenced by *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and PTHREAD_CANCEL_ASYNCCHRONOUS.

56074 The cancelability state and type of any newly created threads, including the thread in which *main()* was first invoked, shall be PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED respectively.

56077 The *pthread_testcancel()* function shall create a cancellation point in the calling thread. The *pthread_testcancel()* function shall have no effect if cancelability is disabled.

56079 **RETURN VALUE**

56080 If successful, the *pthread_setcancelstate()* and *pthread_setcanceltype()* functions shall return zero; otherwise, an error number shall be returned to indicate the error.

56082 **ERRORS**

56083 The *pthread_setcancelstate()* function may fail if:

56084 [EINVAL] The specified state is not PTHREAD_CANCEL_ENABLE or
56085 PTHREAD_CANCEL_DISABLE.

56086 The *pthread_setcanceltype()* function may fail if:

56087 [EINVAL] The specified type is not PTHREAD_CANCEL_DEFERRED or
56088 PTHREAD_CANCEL_ASYNCCHRONOUS.

56089 These functions shall not return an error code of [EINTR].

56090 **EXAMPLES**

56091 None.

56092 **APPLICATION USAGE**

56093 In order to write a signal handler for an asynchronous signal which can run safely in a cancellable thread, *pthread_setcancelstate()* must be used to disable cancellation for the duration of any calls that the signal handler makes which are cancellation points. However, the standard does not permit strictly conforming applications to call *pthread_setcancelstate()* from a signal handler since it is not currently required to be async-signal-safe. On implementations where *pthread_setcancelstate()* is not async-signal-safe, alternatives are to ensure either that the corresponding signals are blocked during execution of functions that are not async-cancel-safe or that cancellation is disabled during times when those signals could be delivered. Implementations are strongly encouraged to make *pthread_setcancelstate()* async-signal-safe.

56102 **RATIONALE**

56103 The *pthread_setcancelstate()* and *pthread_setcanceltype()* functions control the points at which a
56104 thread may be asynchronously canceled. For cancellation control to be usable in modular
56105 fashion, some rules need to be followed.

56106 An object can be considered to be a generalization of a procedure. It is a set of procedures and
56107 global variables written as a unit and called by clients not known by the object. Objects may
56108 depend on other objects.

56109 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On
56110 exit from an object, the cancelability state should always be restored to its value on entry to the
56111 object.

56112 This follows from a modularity argument: if the client of an object (or the client of an object that
56113 uses that object) has disabled cancelability, it is because the client does not want to be concerned
56114 about cleaning up if the thread is canceled while executing some sequence of actions. If an object
56115 is called in such a state and it enables cancelability and a cancellation request is pending for that
56116 thread, then the thread is canceled, contrary to the wish of the client that disabled.

56117 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry
56118 to an object. But as with the cancelability state, on exit from an object the cancelability type
56119 should always be restored to its value on entry to the object.

56120 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously
56121 cancelable.

56122 **FUTURE DIRECTIONS**

56123 The *pthread_setcancelstate()* function may be added to the table of async-signal-safe functions in
56124 [Section 2.4.3](#) (on page 490).

56125 **SEE ALSO**

56126 [*pthread_cancel\(\)*](#)

56127 XBD [`<pthread.h>`](#)

56128 **CHANGE HISTORY**

56129 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

56130 **Issue 6**

56131 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are marked
56132 as part of the Threads option.

56133 **Issue 7**

56134 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are moved
56135 from the Threads option to the Base.

56136 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0294 [622] and XSH/TC2-2008/0295
56137 [615] are applied.

56138 **NAME**

56139 pthread_setconcurrency — set the level of concurrency

56140 **SYNOPSIS**

```
56141 OB XSI #include <pthread.h>  
56142 int pthread_setconcurrency(int new_level);
```

56143 **DESCRIPTION**56144 Refer to *pthread_getconcurrency()*.

56145 **NAME**

56146 pthread_setschedparam ¶ dynamic thread scheduling parameters access (REALTIME
56147 THREADS)

56148 **SYNOPSIS**

```
56149 TPS #include <pthread.h>  
56150 int pthread_setschedparam(pthread_t thread, int policy,  
56151 const struct sched_param *param);
```

56152 **DESCRIPTION**

56153 Refer to [pthread_getschedparam\(\)](#).

56154 **NAME**

56155 pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME**
56156 **THREADS**)

56157 **SYNOPSIS**

```
56158 TPS #include <pthread.h>
56159 int pthread_setschedprio(pthread_t thread, int prio);
```

56160 **DESCRIPTION**

56161 The *pthread_setschedprio()* function shall set the scheduling priority for the thread whose thread
56162 ID is given by *thread* to the value given by *prio*. See [Scheduling Policies](#) (on page 506) for a
56163 description on how this function call affects the ordering of the thread in the thread list for its
56164 new priority.

56165 If the *pthread_setschedprio()* function fails, the scheduling priority of the target thread shall not be
56166 changed.

56167 **RETURN VALUE**

56168 If successful, the *pthread_setschedprio()* function shall return zero; otherwise, an error number
56169 shall be returned to indicate the error.

56170 **ERRORS**

56171 The *pthread_setschedprio()* function may fail if:

- 56172 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.
- 56173 [EPERM] The caller does not have appropriate privileges to set the scheduling priority
56174 of the specified thread.

56175 The *pthread_setschedprio()* function shall not return an error code of [EINTR].

56176 **EXAMPLES**

56177 None.

56178 **APPLICATION USAGE**

56179 None.

56180 **RATIONALE**

56181 The *pthread_setschedprio()* function provides a way for an application to temporarily raise its
56182 priority and then lower it again, without having the undesired side-effect of yielding to other
56183 threads of the same priority. This is necessary if the application is to implement its own
56184 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This
56185 capability is especially important if the implementation does not support the Thread Priority
56186 Protection or Thread Priority Inheritance options, but even if those options are supported it is
56187 needed if the application is to bound priority inheritance for other resources, such as
56188 semaphores.

56189 The standard developers considered that while it might be preferable conceptually to solve this
56190 problem by modifying the specification of *pthread_setschedparam()*, it was too late to make such a
56191 change, as there may be implementations that would need to be changed. Therefore, this new
56192 function was introduced.

56193 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
56194 that the function should fail and report an [ESRCH] error.

56195 **FUTURE DIRECTIONS**

56196 None.

56197 **SEE ALSO**

56198 [Scheduling Policies](#) (on page 506), [pthread_getschedparam\(\)](#)

56199 XBD [<pthread.h>](#)

56200 **CHANGE HISTORY**

56201 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

56202 **Issue 7**

56203 The `pthread_setschedprio()` function is marked only as part of the Thread Execution Scheduling
56204 option as the Threads option is now part of the Base.

56205 Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPERM] error.

56206 Austin Group Interpretation 1003.1-2001 #142 is applied, removing the [ESRCH] error condition.

56207 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0466 [314] is applied.

56208 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0296 [757] is applied.

56209 **NAME**

56210 pthread_setspecific — thread-specific data management

56211 **SYNOPSIS**

56212 #include <pthread.h>

56213 int pthread_setspecific(pthread_key_t *key*, const void **value*);56214 **DESCRIPTION**56215 Refer to *pthread_getspecific()*.

56216 **NAME**

56217 pthread_sigmask, sigprocmask — examine and change blocked signals

56218 **SYNOPSIS**

```
56219 CX #include <signal.h>
56220 int pthread_sigmask(int how, const sigset_t *restrict set,
56221 sigset_t *restrict oset);
56222 int sigprocmask(int how, const sigset_t *restrict set,
56223 sigset_t *restrict oset);
```

56224 **DESCRIPTION**

56225 The *pthread_sigmask()* function shall examine or change (or both) the calling thread's signal
 56226 mask, regardless of the number of threads in the process. The function shall be equivalent to
 56227 *sigprocmask()*, without the restriction that the call be made in a single-threaded process.

56228 In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the
 56229 signal mask of the calling thread.

56230 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the
 56231 currently blocked set.

56232 The argument *how* indicates the way in which the set is changed, and the application shall
 56233 ensure it consists of one of the following values:

56234 SIG_BLOCK The resulting set shall be the union of the current set and the signal set
 56235 pointed to by *set*.

56236 SIG_SETMASK The resulting set shall be the signal set pointed to by *set*.

56237 SIG_UNBLOCK The resulting set shall be the intersection of the current set and the
 56238 complement of the signal set pointed to by *set*.

56239 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location
 56240 pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the
 56241 thread's signal mask shall be unchanged; thus the call can be used to enquire about currently
 56242 blocked signals.

56243 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those
 56244 signals shall be delivered before the call to *sigprocmask()* returns.

56245 It is not possible to block those signals which cannot be ignored. This shall be enforced by the
 56246 system without causing an error to be indicated.

56247 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked,
 56248 the result is undefined, unless the signal was generated by the action of another process, or by
 56249 one of the functions *kill()*, *pthread_kill()*, *raise()*, or *sigqueue()*.

56250 If *sigprocmask()* fails, the thread's signal mask shall not be changed.

56251 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.

56252 **RETURN VALUE**

56253 Upon successful completion *pthread_sigmask()* shall return 0; otherwise, it shall return the
 56254 corresponding error number.

56255 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno*
 56256 shall be set to indicate the error, and the signal mask of the process shall be unchanged.

56257 **ERRORS**56258 The *pthread_sigmask()* and *sigprocmask()* functions shall fail if:56259 [EINVAL] The value of the *how* argument is not equal to one of the defined values.56260 The *pthread_sigmask()* function shall not return an error code of [EINTR].56261 **EXAMPLES**56262 **Signaling in a Multi-Threaded Process**56263 This example shows the use of *pthread_sigmask()* in order to deal with signals in a multi-
56264 threaded process. It provides a fairly general framework that could be easily adapted/extended.

```

56265 #include <stdio.h>
56266 #include <stdlib.h>
56267 #include <pthread.h>
56268 #include <signal.h>
56269 #include <string.h>
56270 #include <errno.h>
56271 ...
56272 static sigset_t  signal_mask; /* signals to block          */
56273 int main (int argc, char *argv[])
56274 {
56275     pthread_t  sig_thr_id;      /* signal handler thread ID */
56276     int        rc;              /* return code               */
56277     sigemptyset (&signal_mask);
56278     sigaddset (&signal_mask, SIGINT);
56279     sigaddset (&signal_mask, SIGTERM);
56280     rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
56281     if (rc != 0) {
56282         /* handle error */
56283         ...
56284     }
56285     /* any newly created threads inherit the signal mask */
56286     rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
56287     if (rc != 0) {
56288         /* handle error */
56289         ...
56290     }
56291     /* APPLICATION CODE */
56292     ...
56293 }
56294 void *signal_thread (void *arg)
56295 {
56296     int        sig_caught;      /* signal caught            */
56297     int        rc;              /* returned code            */
56298     rc = sigwait (&signal_mask, &sig_caught);
56299     if (rc != 0) {
56300         /* handle error */
56301     }

```

```

56302         switch (sig_caught)
56303         {
56304             case SIGINT:      /* process SIGINT */
56305                 ...
56306                 break;
56307             case SIGTERM:    /* process SIGTERM */
56308                 ...
56309                 break;
56310             default:         /* should normally not happen */
56311                 fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
56312                 break;
56313         }
56314     }

```

APPLICATION USAGE

56315 None.

RATIONALE

56318 When a thread's signal mask is changed in a signal-catching function that is installed by
56319 *sigaction()*, the restoration of the signal mask on return from the signal-catching function
56320 overrides that change (see *sigaction()*). If the signal-catching function was installed with
56321 *signal()*, it is unspecified whether this occurs.

56322 See *kill()* for a discussion of the requirement on delivery of signals.

FUTURE DIRECTIONS

56323 None.

SEE ALSO

56326 *exec*, *kill()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*,
56327 *sigpending()*, *sigqueue()*, *sigsuspend()*

56328 XBD <[signal.h](#)>

CHANGE HISTORY

56330 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

56332 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

56333 The *pthread_sigmask()* function is added for alignment with the POSIX Threads Extension.

Issue 6

56334 The *pthread_sigmask()* function is marked as part of the Threads option.

56336 The SYNOPSIS for *sigprocmask()* is marked as a CX extension to note that the presence of this
56337 function in the <[signal.h](#)> header is an extension to the ISO C standard.

56338 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

56339 The DESCRIPTION is updated to explicitly state the functions which may generate the
56340 signal.

56341 The normative text is updated to avoid use of the term “must” for application requirements.

56342 The **restrict** keyword is added to the *pthread_sigmask()* and *sigprocmask()* prototypes for
56343 alignment with the ISO/IEC 9899:1999 standard.

56344 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/105 is applied, updating “process’ signal
56345 mask” to “thread’s signal mask” in the DESCRIPTION and RATIONALE sections.

56346 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/106 is applied, adding the example to the
56347 EXAMPLES section.

56348 **Issue 7**

56349 The *pthread_sigmask()* function is moved from the Threads option to the Base.

56350 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0467 [319] is applied.

56351 **NAME**

56352 pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object

56353 **SYNOPSIS**

56354 #include <pthread.h>

56355 int pthread_spin_destroy(pthread_spinlock_t *lock);

56356 int pthread_spin_init(pthread_spinlock_t *lock, int pshared);

56357 **DESCRIPTION**

56358 The *pthread_spin_destroy()* function shall destroy the spin lock referenced by *lock* and release any
 56359 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is
 56360 reinitialized by another call to *pthread_spin_init()*. The results are undefined if
 56361 *pthread_spin_destroy()* is called when a thread holds the lock, or if this function is called with an
 56362 uninitialized thread spin lock.

56363 The *pthread_spin_init()* function shall allocate any resources required to use the spin lock
 56364 referenced by *lock* and initialize the lock to an unlocked state.

56365 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
 56366 PTHREAD_PROCESS_SHARED, the implementation shall permit the spin lock to be operated
 56367 upon by any thread that has access to the memory where the spin lock is allocated, even if it is
 56368 allocated in memory that is shared by multiple processes.

56369 See [Section 2.9.9](#) (on page 523) for further requirements.

56370 The results are undefined if *pthread_spin_init()* is called specifying an already initialized spin
 56371 lock. The results are undefined if a spin lock is used without first being initialized.

56372 If the *pthread_spin_init()* function fails, the lock is not initialized and the contents of *lock* are
 56373 undefined.

56374 Only the object referenced by *lock* may be used for performing synchronization.

56375 The result of referring to copies of that object in calls to *pthread_spin_destroy()*,
 56376 *pthread_spin_lock()*, *pthread_spin_trylock()*, or *pthread_spin_unlock()* is undefined.

56377 **RETURN VALUE**

56378 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 56379 be returned to indicate the error.

56380 **ERRORS**

56381 The *pthread_spin_init()* function shall fail if:

56382 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

56383 [ENOMEM] Insufficient memory exists to initialize the lock.

56384 These functions shall not return an error code of [EINTR].

56385 **EXAMPLES**

56386 None.

56387 **APPLICATION USAGE**

56388 None.

56389 **RATIONALE**

56390 If an implementation detects that the value specified by the *lock* argument to
 56391 *pthread_spin_destroy()* does not refer to an initialized spin lock object, it is recommended that the
 56392 function should fail and report an [EINVAL] error.

56393 If an implementation detects that the value specified by the *lock* argument to

56394 *pthread_spin_destroy()* or *pthread_spin_init()* refers to a locked spin lock object, or detects that the
56395 value specified by the *lock* argument to *pthread_spin_init()* refers to an already initialized spin
56396 lock object, it is recommended that the function should fail and report an [EBUSY] error.

56397 **FUTURE DIRECTIONS**

56398 None.

56399 **SEE ALSO**

56400 [*pthread_spin_lock\(\)*](#), [*pthread_spin_unlock\(\)*](#)

56401 XBD [**<pthread.h>**](#)

56402 **CHANGE HISTORY**

56403 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

56404 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

56405 **Issue 7**

56406 The *pthread_spin_destroy()* and *pthread_spin_init()* functions are moved from the Spin Locks
56407 option to the Base.

56408 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
56409 undefined behavior.

56410 The [EBUSY] error for a locked spin lock object or an already initialized spin lock object is
56411 removed; this condition results in undefined behavior.

56412 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0297 [972] is applied.

56413 **NAME**

56414 pthread_spin_lock, pthread_spin_trylock — lock a spin lock object

56415 **SYNOPSIS**

56416 #include <pthread.h>

56417 int pthread_spin_lock(pthread_spinlock_t *lock);

56418 int pthread_spin_trylock(pthread_spinlock_t *lock);

56419 **DESCRIPTION**

56420 The *pthread_spin_lock()* function shall lock the spin lock referenced by *lock*. The calling thread
56421 shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is,
56422 shall not return from the *pthread_spin_lock()* call) until the lock becomes available. The results are
56423 undefined if the calling thread holds the lock at the time the call is made. The
56424 *pthread_spin_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any
56425 thread. Otherwise, the function shall fail.

56426 The results are undefined if any of these functions is called with an uninitialized spin lock.

56427 **RETURN VALUE**

56428 Upon successful completion, these functions shall return zero; otherwise, an error number shall
56429 be returned to indicate the error.

56430 **ERRORS**

56431 The *pthread_spin_lock()* function may fail if:

56432 [EDEADLK] A deadlock condition was detected.

56433 The *pthread_spin_trylock()* function shall fail if:

56434 [EBUSY] A thread currently holds the lock.

56435 These functions shall not return an error code of [EINTR].

56436 **EXAMPLES**

56437 None.

56438 **APPLICATION USAGE**

56439 Applications using this function may be subject to priority inversion, as discussed in XBD
56440 [Section 3.291](#) (on page 80).

56441 **RATIONALE**

56442 If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock()*
56443 or *pthread_spin_trylock()* does not refer to an initialized spin lock object, it is recommended that
56444 the function should fail and report an [EINVAL] error.

56445 If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock()*
56446 refers to a spin lock object for which the calling thread already holds the lock, it is recommended
56447 that the function should fail and report an [EDEADLK] error.

56448 **FUTURE DIRECTIONS**

56449 None.

56450 **SEE ALSO**

56451 [pthread_spin_destroy\(\)](#), [pthread_spin_unlock\(\)](#)

56452 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<pthread.h>](#)

56453 **CHANGE HISTORY**

56454 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

56455 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

56456 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/107 is applied, updating the ERRORS
56457 section so that the [EDEADLK] error includes detection of a deadlock condition.

56458 **Issue 7**

56459 The `pthread_spin_lock()` and `pthread_spin_trylock()` functions are moved from the Spin Locks
56460 option to the Base.

56461 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
56462 undefined behavior.

56463 The [EDEADLK] error for a spin lock object for which the calling thread already holds the lock is
56464 removed; this condition results in undefined behavior.

56465 **NAME**

56466 pthread_spin_unlock — unlock a spin lock object

56467 **SYNOPSIS**

56468 #include <pthread.h>

56469 int pthread_spin_unlock(pthread_spinlock_t *lock);

56470 **DESCRIPTION**56471 The *pthread_spin_unlock()* function shall release the spin lock referenced by *lock* which was
56472 locked via the *pthread_spin_lock()* or *pthread_spin_trylock()* functions.

56473 The results are undefined if the lock is not held by the calling thread.

56474 If there are threads spinning on the lock when *pthread_spin_unlock()* is called, the lock becomes
56475 available and an unspecified spinning thread shall acquire the lock.

56476 The results are undefined if this function is called with an uninitialized thread spin lock.

56477 **RETURN VALUE**56478 Upon successful completion, the *pthread_spin_unlock()* function shall return zero; otherwise, an
56479 error number shall be returned to indicate the error.56480 **ERRORS**

56481 This function shall not return an error code of [EINTR].

56482 **EXAMPLES**

56483 None.

56484 **APPLICATION USAGE**

56485 None.

56486 **RATIONALE**56487 If an implementation detects that the value specified by the *lock* argument to
56488 *pthread_spin_unlock()* does not refer to an initialized spin lock object, it is recommended that the
56489 function should fail and report an [EINVAL] error.56490 If an implementation detects that the value specified by the *lock* argument to
56491 *pthread_spin_unlock()* refers to a spin lock object for which the current thread does not hold the
56492 lock, it is recommended that the function should fail and report an [EPERM] error.56493 **FUTURE DIRECTIONS**

56494 None.

56495 **SEE ALSO**56496 [pthread_spin_destroy\(\)](#), [pthread_spin_lock\(\)](#)56497 XBD [Section 4.12](#) (on page 111), [<pthread.h>](#)56498 **CHANGE HISTORY**

56499 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

56500 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.56501 **Issue 7**56502 The *pthread_spin_unlock()* function is moved from the Spin Locks option to the Base.56503 The [EINVAL] error for an uninitialized spin lock object is removed; this condition results in
56504 undefined behavior.

56505
56506

The [EPERM] error for a spin lock object for which the current thread does not hold the lock is removed; this condition results in undefined behavior.

56507 **NAME**

56508 pthread_testcancel — set cancelability state

56509 **SYNOPSIS**

56510 #include <pthread.h>

56511 void pthread_testcancel(void);

56512 **DESCRIPTION**

56513 Refer to *pthread_setcancelstate()*.

56514 **NAME**

56515 ptsname ¶get name of the slave pseudo-terminal device

56516 **SYNOPSIS**

```
56517 XSI #include <stdlib.h>
56518 char *ptsname(int fildev);
```

56519 **DESCRIPTION**

56520 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated
 56521 with a master pseudo-terminal device. The *fildev* argument is a file descriptor that refers to the
 56522 master device. The *ptsname()* function shall return a pointer to a string containing the pathname
 56523 of the corresponding slave device.

56524 The *ptsname()* function need not be thread-safe.

56525 **RETURN VALUE**

56526 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the
 56527 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer and may set
 56528 *errno*. This could occur if *fildev* is an invalid file descriptor or if the slave device name does not
 56529 exist in the file system.

56530 The application shall not modify the string returned. The returned pointer might be invalidated
 56531 or the string content might be overwritten by a subsequent call to *ptsname()*. The returned
 56532 pointer and the string content might also be invalidated if the calling thread is terminated.

56533 **ERRORS**

56534 The *ptsname()* function may fail if:

56535 [EBADF] The *fildev* argument is not a valid file descriptor.

56536 [ENOTTY] The file associated with the *fildev* argument is not a master pseudo-terminal
 56537 device.

56538 **EXAMPLES**

56539 None.

56540 **APPLICATION USAGE**

56541 None.

56542 **RATIONALE**

56543 See the RATIONALE section for *posix_openpt()*.

56544 **FUTURE DIRECTIONS**

56545 None.

56546 **SEE ALSO**

56547 *grantpt()*, *open()*, *posix_openpt()*, *ttyname()*, *unlockpt()*

56548 XBD <stdlib.h>

56549 **CHANGE HISTORY**

56550 First released in Issue 4, Version 2.

56551 **Issue 5**

56552 Moved from X/OPEN UNIX extension to BASE.

56553 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

56554 **Issue 7**

56555 Austin Group Interpretation 1003.1-2001 #156 is applied.

56556 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0468 [75] and XSH/TC1-2008/0469
56557 [96] are applied.

56558 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0298 [503], XSH/TC2-2008/0299 [656],
56559 and XSH/TC2-2008/0300 [503] are applied.

56560 **NAME**

56561 putc — put a byte on a stream

56562 **SYNOPSIS**

56563 #include <stdio.h>

56564 int putc(int *c*, FILE **stream*);56565 **DESCRIPTION**

56566 CX The functionality described on this reference page is aligned with the ISO C standard. Any
56567 conflict between the requirements described here and the ISO C standard is unintentional. This
56568 volume of POSIX.1-2017 defers to the ISO C standard.

56569 The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it
56570 may evaluate *stream* more than once, so the argument should never be an expression with side-
56571 effects.

56572 **RETURN VALUE**56573 Refer to *fputc()*.56574 **ERRORS**56575 Refer to *fputc()*.56576 **EXAMPLES**

56577 None.

56578 **APPLICATION USAGE**

56579 Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side-effects
56580 incorrectly. In particular, *putc(c,*f++)* does not necessarily work correctly. Therefore, use of this
56581 function is not recommended in such situations; *fputc()* should be used instead.

56582 **RATIONALE**

56583 None.

56584 **FUTURE DIRECTIONS**

56585 None.

56586 **SEE ALSO**56587 Section 2.5 (on page 495), *fputc()*

56588 XBD <stdio.h>

56589 **CHANGE HISTORY**

56590 First released in Issue 1. Derived from Issue 1 of the SVID.

56591 **Issue 7**

56592 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0470 [14] is applied.

56593 **NAME**

56594 putc_unlocked ‡'stdio with explicit client locking

56595 **SYNOPSIS**

```
56596 CX       #include <stdio.h>
56597       int putc_unlocked(int c, FILE *stream);
```

56598 **DESCRIPTION**

56599 Refer to [getc_unlocked\(\)](#).

56600 **NAME**

56601 putchar — put a byte on a stdout stream

56602 **SYNOPSIS**

56603 #include <stdio.h>

56604 int putchar(int c);

56605 **DESCRIPTION**

56606 CX The functionality described on this reference page is aligned with the ISO C standard. Any
56607 conflict between the requirements described here and the ISO C standard is unintentional. This
56608 volume of POSIX.1-2017 defers to the ISO C standard.

56609 The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.56610 **RETURN VALUE**56611 Refer to *fputc()*.56612 **ERRORS**56613 Refer to *fputc()*.56614 **EXAMPLES**

56615 None.

56616 **APPLICATION USAGE**

56617 None.

56618 **RATIONALE**

56619 None.

56620 **FUTURE DIRECTIONS**

56621 None.

56622 **SEE ALSO**56623 [Section 2.5](#) (on page 495), *putc()*56624 XBD [<stdio.h>](#)56625 **CHANGE HISTORY**

56626 First released in Issue 1. Derived from Issue 1 of the SVID.

56627 **Issue 7**

56628 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0471 [14] is applied.

56629 **NAME**

56630 putchar_unlocked ¶stdio with explicit client locking

56631 **SYNOPSIS**

56632 CX #include <stdio.h>

56633 int putchar_unlocked(int c);

56634 **DESCRIPTION**

56635 Refer to [getc_unlocked\(\)](#).

56636 **NAME**

56637 putenv — change or add a value to an environment

56638 **SYNOPSIS**

```
56639 XSI      #include <stdlib.h>
56640         int putenv(char *string);
```

56641 **DESCRIPTION**

56642 The *putenv()* function shall use the *string* argument to set environment variable values. The
 56643 *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall
 56644 make the value of the environment variable *name* equal to *value* by altering an existing variable
 56645 or creating a new one. In either case, the string pointed to by *string* shall become part of the
 56646 environment, so altering the string shall change the environment.

56647 The *putenv()* function need not be thread-safe.

56648 **RETURN VALUE**

56649 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value
 56650 and set *errno* to indicate the error.

56651 **ERRORS**

56652 The *putenv()* function may fail if:

56653 [ENOMEM] Insufficient memory was available.

56654 **EXAMPLES**56655 **Changing the Value of an Environment Variable**

56656 The following example changes the value of the *HOME* environment variable to the value
 56657 */usr/home*.

```
56658 #include <stdlib.h>
56659 ...
56660 static char *var = "HOME=/usr/home";
56661 int ret;
56662
56663 ret = putenv(var);
```

56663 **APPLICATION USAGE**

56664 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in
 56665 conjunction with *getenv()*.

56666 See *exec()* for restrictions on changing the environment in multi-threaded applications.

56667 This routine may use *malloc()* to enlarge the environment.

56668 A potential error is to call *putenv()* with an automatic variable as the argument, then return from
 56669 the calling function while *string* is still part of the environment.

56670 Although the space used by *string* is no longer used once a new string which defines *name* is
 56671 passed to *putenv()*, if any thread in the application has used *getenv()* to retrieve a pointer to this
 56672 variable, it should not be freed by calling *free()*. If the changed environment variable is one
 56673 known by the system (such as the locale environment variables) the application should never
 56674 free the buffer used by earlier calls to *putenv()* for the same variable.

56675 The *setenv()* function is preferred over this function. One reason is that *putenv()* is optional and
 56676 therefore less portable. Another is that using *putenv()* can slow down environment searches, as
 56677 explained in the RATIONALE section for *getenv()*.

56678 **RATIONALE**

56679 Refer to the RATIONALE section in *setenv()*.

56680 **FUTURE DIRECTIONS**

56681 None.

56682 **SEE ALSO**

56683 *exec*, *free()*, *getenv()*, *malloc()*, *setenv()*

56684 XBD <stdlib.h>

56685 **CHANGE HISTORY**

56686 First released in Issue 1. Derived from Issue 1 of the SVID.

56687 **Issue 5**

56688 The type of the argument to this function is changed from **const char *** to **char ***. This was
56689 indicated as a FUTURE DIRECTION in previous issues.

56690 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

56691 **Issue 6**

56692 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the
56693 DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to
56694 *exec*.

56695 **Issue 7**

56696 Austin Group Interpretation 1003.1-2001 #156 is applied.

56697 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0472 [167], XSH/TC1-2008/0473 [167],
56698 XSH/TC1-2008/0474 [273,438], and XSH/TC1-2008/0475 [273] are applied.

56699 **NAME**

56700 putmsg, putpmsg ‡send a message on a STREAM\$STREAMS)

56701 **SYNOPSIS**

```
56702 OB XSR #include <stropts.h>
56703 int putmsg(int fildes, const struct strbuf *ctlptr,
56704           const struct strbuf *dataptr, int flags);
56705 int putpmsg(int fildes, const struct strbuf *ctlptr,
56706            const struct strbuf *dataptr, int band, int flags);
```

56707 **DESCRIPTION**

56708 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a
 56709 STREAMS file. The message may contain either a data part, a control part, or both. The data and
 56710 control parts are distinguished by placement in separate buffers, as described below. The
 56711 semantics of each part are defined by the STREAMS module that receives the message.

56712 The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in
 56713 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to
 56714 *putpmsg()*.

56715 The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and
 56716 *dataptr* arguments each point to a **strbuf** structure.

56717 The *ctlptr* argument points to the structure describing the control part, if any, to be included in
 56718 the message. The *buf* member in the **strbuf** structure points to the buffer where the control
 56719 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*
 56720 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if
 56721 any, to be included in the message. The *flags* argument indicates what type of message should be
 56722 sent and is described further below.

56723 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer
 56724 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the
 56725 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part
 56726 shall be sent if either *dataptr(ctlptr)* is a null pointer or the *len* member of *dataptr(ctlptr)* is set to
 56727 -1.

56728 For *putmsg()*, if a control part is specified and *flags* is set to RS_HIPRI, a high priority message
 56729 shall be sent. If no control part is specified, and *flags* is set to RS_HIPRI, *putmsg()* shall fail and
 56730 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be
 56731 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be
 56732 sent and 0 shall be returned.

56733 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following
 56734 mutually-exclusive flags defined: MSG_HIPRI and MSG_BAND. If *flags* is set to 0, *putpmsg()*
 56735 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG_HIPRI
 56736 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG_HIPRI and either
 56737 no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to
 56738 [EINVAL]. If *flags* is set to MSG_BAND, then a message shall be sent in the priority band
 56739 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG_BAND,
 56740 no message shall be sent and 0 shall be returned.

56741 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control
56742 conditions, with the following exceptions:

56743 For high-priority messages, *putmsg()* shall not block on this condition and continues
56744 processing the message.

56745 For other messages, *putmsg()* shall not block but shall fail when the write queue is full and
56746 O_NONBLOCK is set.

56747 The *putmsg()* function shall also block, unless prevented by lack of internal resources, while
56748 waiting for the availability of message blocks in the STREAM, regardless of priority or whether
56749 O_NONBLOCK has been specified. No partial message shall be sent.

56750 RETURN VALUE

56751 Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return
56752 -1 and set *errno* to indicate the error.

56753 ERRORS

56754 The *putmsg()* and *putpmsg()* functions shall fail if:

56755 [EAGAIN] A non-priority message was specified, the O_NONBLOCK flag is set, and the
56756 STREAM write queue is full due to internal flow control conditions; or buffers
56757 could not be allocated for the message that was to be created.

56758 [EBADF] *fildes* is not a valid file descriptor open for writing.

56759 [EINTR] A signal was caught during *putmsg()*.

56760 [EINVAL] An undefined value is specified in *flags*, or *flags* is set to RS_HIPRI or
56761 MSG_HIPRI and no control part is supplied, or the STREAM or multiplexer
56762 referenced by *fildes* is linked (directly or indirectly) downstream from a
56763 multiplexer, or *flags* is set to MSG_HIPRI and *band* is non-zero (for *putpmsg()*
56764 only).

56765 [ENOSR] Buffers could not be allocated for the message that was to be created due to
56766 insufficient STREAMS memory resources.

56767 [ENOSTR] A STREAM is not associated with *fildes*.

56768 [ENXIO] A hangup condition was generated downstream for the specified STREAM.

56769 [EPIPE] or [EIO] The *fildes* argument refers to a STREAMS-based pipe and the other end of the
56770 pipe is closed. A SIGPIPE signal is generated for the calling thread.

56771 [ERANGE] The size of the data part of the message does not fall within the range
56772 specified by the maximum and minimum packet sizes of the topmost
56773 STREAM module. This value is also returned if the control part of the message
56774 is larger than the maximum configured size of the control part of a message,
56775 or if the data part of a message is larger than the maximum configured size of
56776 the data part of a message.

56777 In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an
56778 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
56779 *putmsg()* or *putpmsg()*, but reflects the prior error.

56780 **EXAMPLES**56781 **Sending a High-Priority Message**

56782 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the
56783 following:

- 56784 1. Creates a high-priority message with a control part and a data part, using the buffers
56785 pointed to by *ctrlbuf* and *databuf*, respectively.
- 56786 2. Sends the message to the STREAMS file identified by *fd*.

```
56787 #include <stropts.h>
56788 #include <string.h>
56789 ...
56790 int fd;
56791 char *ctrlbuf = "This is the control part";
56792 char *databuf = "This is the data part";
56793 struct strbuf ctrl;
56794 struct strbuf data;
56795 int ret;

56796 ctrl.buf = ctrlbuf;
56797 ctrl.len = strlen(ctrlbuf);

56798 data.buf = databuf;
56799 data.len = strlen(databuf);

56800 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

56801 **Using putpmsg()**

56802 This example has the same effect as the previous example. In this example, however, the
56803 *putpmsg()* function creates and sends the message to the STREAMS file.

```
56804 #include <stropts.h>
56805 #include <string.h>
56806 ...
56807 int fd;
56808 char *ctrlbuf = "This is the control part";
56809 char *databuf = "This is the data part";
56810 struct strbuf ctrl;
56811 struct strbuf data;
56812 int ret;

56813 ctrl.buf = ctrlbuf;
56814 ctrl.len = strlen(ctrlbuf);

56815 data.buf = databuf;
56816 data.len = strlen(databuf);

56817 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

56818 **APPLICATION USAGE**

56819 None.

56820 **RATIONALE**

56821 None.

56822 **FUTURE DIRECTIONS**56823 The *putmsg()* and *putpmsg()* functions may be removed in a future version.56824 **SEE ALSO**56825 [Section 2.6](#) (on page 500), *getmsg()*, *poll()*, *read()*, *write()*56826 XBD [<stropts.h>](#)56827 **CHANGE HISTORY**

56828 First released in Issue 4, Version 2.

56829 **Issue 5**

56830 Moved from X/OPEN UNIX extension to BASE.

56831 The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the
56832 control part of a message generated by *putmsg()* is at least 64 bytes in length”.56833 **Issue 6**

56834 This function is marked as part of the XSI STREAMS Option Group.

56835 The normative text is updated to avoid use of the term “must” for application requirements.

56836 **Issue 7**56837 The *putmsg()* and *putpmsg()* functions are marked obsolescent.

56838 **NAME**

56839 puts ¶put a string on standard output

56840 **SYNOPSIS**

```
56841 #include <stdio.h>
56842 int puts(const char *s);
```

56843 **DESCRIPTION**

56844 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56845 conflict between the requirements described here and the ISO C standard is unintentional. This
 56846 volume of POSIX.1-2017 defers to the ISO C standard.

56847 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the
 56848 standard output stream *stdout*. The terminating null byte shall not be written.

56849 CX The last data modification and last file status change timestamps of the file shall be marked for
 56850 update between the successful execution of *puts()* and the next successful completion of a call to
 56851 *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

56852 **RETURN VALUE**

56853 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall
 56854 CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

56855 **ERRORS**56856 Refer to *fputc()*.56857 **EXAMPLES**56858 **Printing to Standard Output**

56859 The following example gets the current time, converts it to a string using *localtime()* and
 56860 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to
 56861 an event for which it is waiting.

```
56862 #include <time.h>
56863 #include <stdio.h>
56864 ...
56865 time_t now;
56866 int minutes_to_event;
56867 ...
56868 time(&now);
56869 printf("The time is ");
56870 puts(asctime(localtime(&now)));
56871 printf("There are %d minutes to the event.\n",
56872     minutes_to_event);
56873 ...
```

56874 **APPLICATION USAGE**56875 The *puts()* function appends a <newline>, while *fputs()* does not.

56876 This volume of POSIX.1-2017 requires that successful completion simply return a non-negative
 56877 integer. There are at least three known different implementation conventions for this
 56878 requirement:

56879 Return a constant value.

56880 Return the last character written.

56881 Return the number of bytes written. Note that this implementation convention cannot be
56882 adhered to for strings longer than {INT_MAX} bytes as the value would not be
56883 representable in the return type of the function. For backwards compatibility,
56884 implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and
56885 return {INT_MAX} for all longer strings.

56886 **RATIONALE**

56887 None.

56888 **FUTURE DIRECTIONS**

56889 None.

56890 **SEE ALSO**

56891 [Section 2.5](#) (on page 495), [fopen\(\)](#), [fputs\(\)](#), [putc\(\)](#)

56892 XBD [<stdio.h>](#)

56893 **CHANGE HISTORY**

56894 First released in Issue 1. Derived from Issue 1 of the SVID.

56895 **Issue 6**

56896 Extensions beyond the ISO C standard are marked.

56897 **Issue 7**

56898 Changes are made related to support for finegrained timestamps.

56899 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0476 [174,412] and
56900 XSH/TC1-2008/0477 [14] are applied.

56901 **NAME**

56902 pututxline ¶put an entry into the user accounting database

56903 **SYNOPSIS**

```
56904 XSI       #include <utmpx.h>  
56905       struct utmpx *pututxline(const struct utmpx *utmpx);
```

56906 **DESCRIPTION**56907 Refer to *endutxent()*.

56908 **NAME**

56909 putwc — put a wide character on a stream

56910 **SYNOPSIS**

56911 #include <stdio.h>

56912 #include <wchar.h>

56913 wint_t putwc(wchar_t *wc*, FILE **stream*);56914 **DESCRIPTION**

56915 CX The functionality described on this reference page is aligned with the ISO C standard. Any
56916 conflict between the requirements described here and the ISO C standard is unintentional. This
56917 volume of POSIX.1-2017 defers to the ISO C standard.

56918 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro
56919 it may evaluate *stream* more than once, so the argument should never be an expression with
56920 side-effects.

56921 **RETURN VALUE**56922 Refer to *fputwc()*.56923 **ERRORS**56924 Refer to *fputwc()*.56925 **EXAMPLES**

56926 None.

56927 **APPLICATION USAGE**

56928 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side-effects
56929 incorrectly. In particular, *putwc(wc,*f++)* need not work correctly. Therefore, use of this function
56930 is not recommended; *fputwc()* should be used instead.

56931 **RATIONALE**

56932 None.

56933 **FUTURE DIRECTIONS**

56934 None.

56935 **SEE ALSO**56936 [Section 2.5](#) (on page 495), *fputwc()*56937 XBD [<stdio.h>](#), [<wchar.h>](#)56938 **CHANGE HISTORY**

56939 First released as a World-wide Portability Interface in Issue 4.

56940 **Issue 5**

56941 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
56942 is changed from **wint_t** to **wchar_t**.

56943 The Optional Header (OH) marking is removed from [<stdio.h>](#).56944 **Issue 7**

56945 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0478 [14] is applied.

56946 **NAME**

56947 putwchar — put a wide character on a stdout stream

56948 **SYNOPSIS**

56949 #include <wchar.h>

56950 wint_t putwchar(wchar_t wc);

56951 **DESCRIPTION**

56952 CX The functionality described on this reference page is aligned with the ISO C standard. Any
56953 conflict between the requirements described here and the ISO C standard is unintentional. This
56954 volume of POSIX.1-2017 defers to the ISO C standard.

56955 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.56956 **RETURN VALUE**56957 Refer to *fputwc()*.56958 **ERRORS**56959 Refer to *fputwc()*.56960 **EXAMPLES**

56961 None.

56962 **APPLICATION USAGE**

56963 None.

56964 **RATIONALE**

56965 None.

56966 **FUTURE DIRECTIONS**

56967 None.

56968 **SEE ALSO**56969 [Section 2.5](#) (on page 495), *fputwc()*, *putwc()*56970 XBD [<wchar.h>](#)56971 **CHANGE HISTORY**

56972 First released in Issue 4.

56973 **Issue 5**56974 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
56975 is changed from **wint_t** to **wchar_t**.56976 **Issue 7**

56977 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0479 [14] is applied.

56978 **NAME**

56979 pwrite ¶write on a file

56980 **SYNOPSIS**

56981 #include <unistd.h>

56982 ssize_t pwrite(int *fd*, const void **buf*, size_t *nbyte*,
56983 off_t *offset*);

56984 **DESCRIPTION**

56985 Refer to *write()*.

56986 **NAME**

56987 qsort ‡'sort a table of data

56988 **SYNOPSIS**

56989 #include <stdlib.h>

56990 void qsort(void *base, size_t nel, size_t width,
56991 int (*compar)(const void *, const void *));56992 **DESCRIPTION**56993 CX The functionality described on this reference page is aligned with the ISO C standard. Any
56994 conflict between the requirements described here and the ISO C standard is unintentional. This
56995 volume of POSIX.1-2017 defers to the ISO C standard.56996 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by
56997 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has
56998 the value zero, the comparison function pointed to by *compar* shall not be called and no
56999 rearrangement shall take place.57000 The application shall ensure that the comparison function pointed to by *compar* does not alter the
57001 contents of the array. The implementation may reorder elements of the array between calls to the
57002 comparison function, but shall not alter the contents of any individual element.57003 When the same objects (consisting of *width* bytes, irrespective of their current positions in the
57004 array) are passed more than once to the comparison function, the results shall be consistent with
57005 one another. That is, they shall define a total ordering on the array.57006 The contents of the array shall be sorted in ascending order according to a comparison function.
57007 The *compar* argument is a pointer to the comparison function, which is called with two
57008 arguments that point to the elements being compared. The application shall ensure that the
57009 function returns an integer less than, equal to, or greater than 0, if the first argument is
57010 considered respectively less than, equal to, or greater than the second. If two members compare
57011 as equal, their order in the sorted array is unspecified.57012 **RETURN VALUE**57013 The *qsort()* function shall not return a value.57014 **ERRORS**

57015 No errors are defined.

57016 **EXAMPLES**

57017 None.

57018 **APPLICATION USAGE**57019 The comparison function need not compare every byte, so arbitrary data may be contained in
57020 the elements in addition to the values being compared.57021 **RATIONALE**57022 The requirement that each argument (hereafter referred to as *p*) to the comparison function is a
57023 pointer to elements of the array implies that for every call, for each argument separately, all of
57024 the following expressions are non-zero:57025 ((char *)p - (char *)base) % width == 0
57026 (char *)p >= (char *)base
57027 (char *)p < (char *)base + nel * width

57028 **FUTURE DIRECTIONS**

57029 None.

57030 **SEE ALSO**57031 *alphasort()*

57032 XBD <stdlib.h>

57033 **CHANGE HISTORY**

57034 First released in Issue 1. Derived from Issue 1 of the SVID.

57035 **Issue 6**

57036 The normative text is updated to avoid use of the term “must” for application requirements.

57037 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to
57038 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The
57039 RATIONALE is also updated. These changes are for alignment with the ISO C standard.

57040 **NAME**

57041 raise — send a signal to the executing process

57042 **SYNOPSIS**

57043 #include <signal.h>

57044 int raise(int sig);

57045 **DESCRIPTION**

57046 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 57047 conflict between the requirements described here and the ISO C standard is unintentional. This
 57048 volume of POSIX.1-2017 defers to the ISO C standard.

57049 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal
 57050 handler is called, the *raise()* function shall not return until after the signal handler does.

57051 CX The effect of the *raise()* function shall be equivalent to calling:

57052 pthread_kill(pthread_self(), sig);

57053 **RETURN VALUE**

57054 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned
 57055 and *errno* shall be set to indicate the error.

57056 **ERRORS**57057 The *raise()* function shall fail if:

57058 CX [EINVAL] The value of the *sig* argument is an invalid signal number.

57059 **EXAMPLES**

57060 None.

57061 **APPLICATION USAGE**

57062 None.

57063 **RATIONALE**

57064 The term “thread” is an extension to the ISO C standard.

57065 **FUTURE DIRECTIONS**

57066 None.

57067 **SEE ALSO**57068 *kill()*, *sigaction()*

57069 XBD <signal.h>, <sys/types.h>

57070 **CHANGE HISTORY**

57071 First released in Issue 4. Derived from the ANSI C standard.

57072 **Issue 5**

57073 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

57074 **Issue 6**

57075 Extensions beyond the ISO C standard are marked.

57076 The following new requirements on POSIX implementations derive from alignment with the
 57077 Single UNIX Specification:

57078 In the RETURN VALUE section, the requirement to set *errno* on error is added.

57079

The [EINVAL] error condition is added.

57080 **Issue 7**

57081

Functionality relating to the Threads option is moved to the Base.

57082 **NAME**

57083 rand, rand_r, srand ‡pseudo-random number generator

57084 **SYNOPSIS**

```
57085 #include <stdlib.h>
57086 int rand(void);
57087 OB CX int rand_r(unsigned *seed);
57088 void srand(unsigned seed);
```

57089 **DESCRIPTION**

57090 CX For *rand()* and *srand()*: The functionality described on this reference page is aligned with the
 57091 ISO C standard. Any conflict between the requirements described here and the ISO C standard is
 57092 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

57093 The *rand()* function shall compute a sequence of pseudo-random integers in the range
 57094 XSI [0,{RAND_MAX}] with a period of at least 2^{32} .

57095 CX The *rand()* function need not be thread-safe.

57096 OB CX The *rand_r()* function shall compute a sequence of pseudo-random integers in the range
 57097 [0,{RAND_MAX}]. (The value of the {RAND_MAX} macro shall be at least 32767.)

57098 If *rand_r()* is called with the same initial value for the object pointed to by *seed* and that object is
 57099 not modified between successive returns and calls to *rand_r()*, the same sequence shall be
 57100 generated.

57101 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random
 57102 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same
 57103 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before
 57104 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first
 57105 called with a seed value of 1.

57106 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 57107 *rand()* or *srand()*.

57108 **RETURN VALUE**

57109 The *rand()* function shall return the next pseudo-random number in the sequence.

57110 OB CX The *rand_r()* function shall return a pseudo-random integer.

57111 The *srand()* function shall not return a value.

57112 **ERRORS**

57113 No errors are defined.

57114 **EXAMPLES**57115 **Generating a Pseudo-Random Number Sequence**

57116 The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
57117 #include <stdio.h>
57118 #include <stdlib.h>
57119 ...
57120     long count, i;
57121     char *keyst;
57122     int elementlen, len;
57123     char c;
57124     ...
```

```

57125     /* Initial random number generator. */
57126         srand(1);

57127     /* Create keys using only lowercase characters */
57128     len = 0;
57129     for (i=0; i<count; i++) {
57130         while (len < elementlen) {
57131             c = (char) (rand() % 128);
57132             if (islower(c))
57133                 keystr[len++] = c;
57134         }

57135         keystr[len] = '\0';
57136         printf("%s Element%0*ld\n", keystr, elementlen, i);
57137         len = 0;
57138     }

```

57139 Generating the Same Sequence on Different Machines

57140 The following code defines a pair of functions that could be incorporated into applications
57141 wishing to ensure that the same sequence of numbers is generated across different machines.

```

57142     static unsigned long next = 1;
57143     int myrand(void) /* RAND_MAX assumed to be 32767. */
57144     {
57145         next = next * 1103515245 + 12345;
57146         return((unsigned)(next/65536) % 32768);
57147     }

57148     void mysrand(unsigned seed)
57149     {
57150         next = seed;
57151     }

```

57152 APPLICATION USAGE

57153 The *drand48()* and *random()* functions provide much more elaborate pseudo-random number
57154 generators.

57155 The limitations on the amount of state that can be carried between one function call and another
57156 mean the *rand_r()* function can never be implemented in a way which satisfies all of the
57157 requirements on a pseudo-random number generator.

57158 These functions should be avoided whenever non-trivial requirements (including safety) have to
57159 be fulfilled.

57160 RATIONALE

57161 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams
57162 shared by all threads. Those two functions need not change, but there has to be mutual-
57163 exclusion that prevents interference between two threads concurrently accessing the random
57164 number generator.

57165 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded
57166 program:

- 57167 1. A single per-process sequence of pseudo-random numbers that is shared by all threads
57168 that call *rand()*

- 57169 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*
- 57170 This is provided by the modified thread-safe function based on whether the seed value is global
57171 to the entire process or local to each thread.
- 57172 This does not address the known deficiencies of the *rand()* function implementations, which
57173 have been approached by maintaining more state. In effect, this specifies new thread-safe forms
57174 of a deficient function.
- 57175 **FUTURE DIRECTIONS**
- 57176 The *rand_r()* function may be removed in a future version.
- 57177 **SEE ALSO**
- 57178 *drand48()*, *initstate()*
- 57179 XBD <stdlib.h>
- 57180 **CHANGE HISTORY**
- 57181 First released in Issue 1. Derived from Issue 1 of the SVID.
- 57182 **Issue 5**
- 57183 The *rand_r()* function is included for alignment with the POSIX Threads Extension.
- 57184 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.
- 57185 **Issue 6**
- 57186 Extensions beyond the ISO C standard are marked.
- 57187 The *rand_r()* function is marked as part of the Thread-Safe Functions option.
- 57188 **Issue 7**
- 57189 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 57190 The *rand_r()* function is marked obsolescent.
- 57191 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0301 [743] is applied.

57192 **NAME**

57193 random ‡generate pseudo-random number

57194 **SYNOPSIS**

57195 XSI #include <stdlib.h>

57196 long random(void);

57197 **DESCRIPTION**

57198 Refer to *initstate()*.

57199 **NAME**

57200 pread, read — read from a file

57201 **SYNOPSIS**

57202 #include <unistd.h>

57203 ssize_t pread(int *fildes*, void *buf, size_t *nbyte*, off_t *offset*);57204 ssize_t read(int *fildes*, void *buf, size_t *nbyte*);57205 **DESCRIPTION**

57206 The *read()* function shall attempt to read *nbyte* bytes from the file associated with the open file
 57207 descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on
 57208 the same pipe, FIFO, or terminal device is unspecified.

57209 Before any action described below is taken, and if *nbyte* is zero, the *read()* function may detect
 57210 and return errors as described below. In the absence of errors, or if error detection is not
 57211 performed, the *read()* function shall return zero and have no other results.

57212 On files that support seeking (for example, a regular file), the *read()* shall start at a position in
 57213 the file given by the file offset associated with *fildes*. The file offset shall be incremented by the
 57214 number of bytes actually read.

57215 Files that do not support seeking—for example, terminals—always read from the current
 57216 position. The value of a file offset associated with such a file is undefined.

57217 No data transfer shall occur past the current end-of-file. If the starting position is at or after the
 57218 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent
 57219 *read()* requests is implementation-defined.

57220 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

57221 When attempting to read from an empty pipe or FIFO:

57222 If no process has the pipe open for writing, *read()* shall return 0 to indicate end-of-file.

57223 If some process has the pipe open for writing and O_NONBLOCK is set, *read()* shall return
 57224 -1 and set *errno* to [EAGAIN].

57225 If some process has the pipe open for writing and O_NONBLOCK is clear, *read()* shall
 57226 block the calling thread until some data is written or the pipe is closed by all processes that
 57227 had the pipe open for writing.

57228 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and
 57229 has no data currently available:

57230 If O_NONBLOCK is set, *read()* shall return -1 and set *errno* to [EAGAIN].

57231 If O_NONBLOCK is clear, *read()* shall block the calling thread until some data becomes
 57232 available.

57233 The use of the O_NONBLOCK flag has no effect if there is some data available.

57234 The *read()* function reads data previously written to a file. If any portion of a regular file prior to
 57235 the end-of-file has not been written, *read()* shall return bytes with value 0. For example, *lseek()*
 57236 allows the file offset to be set beyond the end of existing data in the file. If data is later written at
 57237 this point, subsequent reads in the gap between the previous end of data and the newly written
 57238 data shall return bytes with value 0 until data is written into the gap.

57239 Upon successful completion, where *nbyte* is greater than 0, *read()* shall mark for update the last
 57240 data access timestamp of the file, and shall return the number of bytes read. This number shall
 57241 never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes

57242		left in the file is less than <i>nbyte</i> , if the <i>read()</i> request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than <i>nbyte</i> bytes immediately available for reading.
57243		For example, a <i>read()</i> from a file associated with a terminal may return one typed line of data.
57244		
57245		If a <i>read()</i> is interrupted by a signal before it reads any data, it shall return -1 with <i>errno</i> set to
57246		[EINTR].
57247		If a <i>read()</i> is interrupted by a signal after it has successfully read some data, it shall return the
57248		number of bytes read.
57249		For regular files, no data transfer shall occur past the offset maximum established in the open
57250		file description associated with <i>fildes</i> .
57251		If <i>fildes</i> refers to a socket, <i>read()</i> shall be equivalent to <i>recv()</i> with no flags set.
57252	SIO	If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor
57253		shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and
57254		O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as
57255		defined by synchronized I/O file integrity completion.
57256	SHM	If <i>fildes</i> refers to a shared memory object, the result of the <i>read()</i> function is unspecified.
57257	TYM	If <i>fildes</i> refers to a typed memory object, the result of the <i>read()</i> function is unspecified.
57258	OB XSR	A <i>read()</i> from a STREAMS file can read data in three different modes: <i>byte-stream</i> mode, <i>message-</i>
57259		<i>nondiscard</i> mode, and <i>message-discard</i> mode. The default shall be <i>byte-stream</i> mode. This can be
57260		changed using the I_SRDOPT <i>ioctl()</i> request, and can be tested with I_GRDOPT <i>ioctl()</i> . In <i>byte-</i>
57261		<i>stream</i> mode, <i>read()</i> shall retrieve data from the STREAM until as many bytes as were requested
57262		are transferred, or until there is no more data to be retrieved. <i>Byte-stream</i> mode ignores
57263		message boundaries.
57264		In STREAMS <i>message-nondiscard</i> mode, <i>read()</i> shall retrieve data until as many bytes as were
57265		requested are transferred, or until a message boundary is reached. If <i>read()</i> does not retrieve all
57266		the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by
57267		the next <i>read()</i> call. <i>Message-discard</i> mode also retrieves data until as many bytes as were
57268		requested are transferred, or a message boundary is reached. However, unread data remaining
57269		in a message after the <i>read()</i> returns shall be discarded, and shall not be available for a
57270		subsequent <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> call.
57271		How <i>read()</i> handles zero-byte STREAMS messages is determined by the current read mode
57272		setting. In <i>byte-stream</i> mode, <i>read()</i> shall accept data until it has read <i>nbyte</i> bytes, or until there
57273		is no more data to read, or until a zero-byte message block is encountered. The <i>read()</i> function
57274		shall then return the number of bytes read, and place the zero-byte message back on the
57275		STREAM to be retrieved by the next <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> . In <i>message-nondiscard</i>
57276		mode or <i>message-discard</i> mode, a zero-byte message shall return 0 and the message shall be
57277		removed from the STREAM. When a zero-byte message is read as the first message on a
57278		STREAM, the message shall be removed from the STREAM and 0 shall be returned, regardless
57279		of the read mode.
57280		A <i>read()</i> from a STREAMS file shall return the data in the message at the front of the STREAM
57281		head read queue, regardless of the priority band of the message.
57282		By default, STREAMs are in control-normal mode, in which a <i>read()</i> from a STREAMS file can
57283		only process messages that contain a data part but do not contain a control part. The <i>read()</i> shall
57284		fail if a message containing a control part is encountered at the STREAM head. This default
57285		action can be changed by placing the STREAM in either control-data mode or control-discard
57286		mode with the I_SRDOPT <i>ioctl()</i> command. In control-data mode, <i>read()</i> shall convert any
57287		control part to data and pass it to the application before passing any data part originally present

57288 in the same message. In control-discard mode, *read()* shall discard message control parts but
57289 return to the process any data part in the message.

57290 In addition, *read()* shall fail if the STREAM head had processed an asynchronous error before
57291 the call. In this case, the value of *errno* shall not reflect the result of *read()*, but reflect the prior
57292 error. If a hangup occurs on the STREAM being read, *read()* shall continue to operate normally
57293 until the STREAM head read queue is empty. Thereafter, it shall return 0.

57294 The *pread()* function shall be equivalent to *read()*, except that it shall read from a given position
57295 in the file without changing the file offset. The first three arguments to *pread()* are the same as
57296 *read()* with the addition of a fourth argument *offset* for the desired position inside the file. An
57297 attempt to perform a *pread()* on a file that is incapable of seeking shall result in an error.

57298 RETURN VALUE

57299 Upon successful completion, these functions shall return a non-negative integer indicating the
57300 number of bytes actually read. Otherwise, the functions shall return -1 and set *errno* to indicate
57301 the error.

57302 ERRORS

57303 These functions shall fail if:

57304 [EAGAIN] The file is neither a pipe, nor a FIFO, nor a socket, the O_NONBLOCK flag is
57305 set for the file descriptor, and the thread would be delayed in the read
57306 operation.

57307 [EBADF] The *fildev* argument is not a valid file descriptor open for reading.

57308 OB XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message
57309 waiting to be read includes a control part.

57310 [EINTR] The read operation was terminated due to the receipt of a signal, and no data
57311 was transferred.

57312 OB XSR [EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or
57313 indirectly) downstream from a multiplexer.

57314 [EIO] The process is a member of a background process group attempting to read
57315 from its controlling terminal, and either the calling thread is blocking
57316 SIGTTIN or the process is ignoring SIGTTIN or the process group of the
57317 process is orphaned. This error may also be generated for implementation-
57318 defined reasons.

57319 XSI [EISDIR] The *fildev* argument refers to a directory and the implementation does not
57320 allow the directory to be read using *read()* or *pread()*. The *readdir()* function
57321 should be used instead.

57322 [EOVERFLOW] The file is a regular file, *nbyte* is greater than 0, the starting position is before
57323 the end-of-file, and the starting position is greater than or equal to the offset
57324 maximum established in the open file description associated with *fildev*.

57325 The *pread()* function shall fail if:

57326 [EINVAL] The file is a regular file or block special file, and the *offset* argument is
57327 negative. The file offset shall remain unchanged.

57328 [ESPIPE] The file is incapable of seeking.

- 57329 The `read()` function shall fail if:
- 57330 [EAGAIN] The file is a pipe or FIFO, the `O_NONBLOCK` flag is set for the file descriptor,
57331 and the thread would be delayed in the read operation.
- 57332 [EAGAIN] or [EWOULDBLOCK]
57333 The file is a socket, the `O_NONBLOCK` flag is set for the file descriptor, and
57334 the thread would be delayed in the read operation.
- 57335 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by
57336 its peer.
- 57337 [ENOTCONN] A read was attempted on a socket that is not connected.
- 57338 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.
- 57339 These functions may fail if:
- 57340 [EIO] A physical I/O error has occurred.
- 57341 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 57342 [ENOMEM] Insufficient memory was available to fulfill the request.
- 57343 [ENXIO] A request was made of a nonexistent device, or the request was outside the
57344 capabilities of the device.

57345 EXAMPLES

57346 Reading Data into a Buffer

57347 The following example reads data from the file associated with the file descriptor `fd` into the
57348 buffer pointed to by `buf`.

```
57349 #include <sys/types.h>
57350 #include <unistd.h>
57351 ...
57352 char buf[20];
57353 size_t nbytes;
57354 ssize_t bytes_read;
57355 int fd;
57356 ...
57357 nbytes = sizeof(buf);
57358 bytes_read = read(fd, buf, nbytes);
57359 ...
```

57360 APPLICATION USAGE

57361 None.

57362 RATIONALE

57363 This volume of POSIX.1-2017 does not specify the value of the file offset after an error is
57364 returned; there are too many cases. For programming errors, such as [EBADF], the concept is
57365 meaningless since no file is involved. For errors that are detected immediately, such as
57366 [EAGAIN], clearly the offset should not change. After an interrupt or hardware error, however,
57367 an updated value would be very useful and is the behavior of many implementations.

57368 Note that a `read()` of zero bytes does not modify the last data access timestamp. A `read()` that
57369 requests more than zero bytes, but returns zero, is required to modify the last data access
57370 timestamp.

57371 Implementations are allowed, but not required, to perform error checking for *read()* requests of
57372 zero bytes.

57373 **Input and Output**

57374 The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and
57375 *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use
57376 abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be
57377 declared so that existing functions work, but can also be declared so that larger types can be
57378 represented in future implementations. It is presumed that whatever constraints limit the
57379 maximum range of **size_t** also limit portable I/O requests to the same range. This volume of
57380 POSIX.1-2017 also limits the range further by requiring that the byte count be limited so that a
57381 signed return value remains meaningful. Since the return type is also a (signed) abstract type,
57382 the byte count can be defined by the implementation to be larger than an **int** can hold.

57383 The standard developers considered adding atomicity requirements to a pipe or FIFO, but
57384 recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of
57385 reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

57386 This volume of POSIX.1-2017 requires that no action be taken for *read()* or *write()* when *nbyte* is
57387 zero. This is not intended to take precedence over detection of errors (such as invalid buffer
57388 pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-2017, but
57389 the phrasing here could be misread to require detection of the zero case before any other errors.
57390 A value of zero is to be considered a correct value, for which the semantics are a no-op.

57391 I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the
57392 bytes from a single operation that started out together end up together, without interleaving
57393 from other I/O operations. It is a known attribute of terminals that this is not honored, and
57394 terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified.
57395 The behavior for other device types is also left unspecified, but the wording is intended to imply
57396 that future standards might choose to specify atomicity (or not).

57397 There were recommendations to add format parameters to *read()* and *write()* in order to handle
57398 networked transfers among heterogeneous file system and base hardware types. Such a facility
57399 may be required for support by the OSI presentation of layer services. However, it was
57400 determined that this should correspond with similar C-language facilities, and that is beyond
57401 the scope of this volume of POSIX.1-2017. The concept was suggested to the developers of the
57402 ISO C standard for their consideration as a possible area for future work.

57403 In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does
57404 not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth
57405 Edition, there is an additional function, *select()*, whose purpose is to pause until specified
57406 activity (data to read, space to write, and so on) is detected on specified file descriptors. It is
57407 common in applications written for those systems for *select()* to be used before *read()* in
57408 situations (such as keyboard input) where interruption of I/O due to a signal is desired.

57409 The issue of which files or file types are interruptible is considered an implementation design
57410 issue. This is often affected primarily by hardware and reliability issues.

57411 There are no references to actions taken following an "unrecoverable error". It is considered
57412 beyond the scope of this volume of POSIX.1-2017 to describe what happens in the case of
57413 hardware errors.

57414 Earlier versions of this standard allowed two very different behaviors with regard to the
57415 handling of interrupts. In order to minimize the resulting confusion, it was decided that
57416 POSIX.1-2017 should support only one of these behaviors. Historical practice on AT&T-derived

57417 systems was to have *read()* and *write()* return `-1` and set *errno* to `[EINTR]` when interrupted after
 57418 some, but not all, of the data requested had been transferred. However, the US Department of
 57419 Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read()* and
 57420 *write()* return the number of bytes actually transferred before the interrupt. If `-1` is returned
 57421 when any data is transferred, it is difficult to recover from the error on a seekable device and
 57422 impossible on a non-seekable device. Most new implementations support this behavior. The
 57423 behavior required by POSIX.1-2017 is to return the number of bytes transferred.

57424 POSIX.1-2017 does not specify when an implementation that buffers *read()*s actually moves the
 57425 data into the user-supplied buffer, so an implementation may choose to do this at the latest
 57426 possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a
 57427 partial byte count, but rather to return `-1` and set *errno* to `[EINTR]`.

57428 Consideration was also given to combining the two previous options, and setting *errno* to
 57429 `[EINTR]` while returning a short count. However, not only is there no existing practice that
 57430 implements this, it is also contradictory to the idea that when *errno* is set, the function
 57431 responsible shall return `-1`.

57432 This volume of POSIX.1-2017 intentionally does not specify any *pread()* errors related to pipes,
 57433 FIFOs, and sockets other than `[ESPIPE]`.

57434 FUTURE DIRECTIONS

57435 None.

57436 SEE ALSO

57437 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*

57438 XBD Chapter 11 (on page 199), `<stropts.h>`, `<sys/uio.h>`, `<unistd.h>`

57439 CHANGE HISTORY

57440 First released in Issue 1. Derived from Issue 1 of the SVID.

57441 Issue 5

57442 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 57443 Threads Extension.

57444 Large File Summit extensions are added.

57445 The *pread()* function is added.

57446 Issue 6

57447 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
 57448 marked as part of the XSI STREAMS Option Group.

57449 The following new requirements on POSIX implementations derive from alignment with the
 57450 Single UNIX Specification:

57451 The DESCRIPTION now states that if *read()* is interrupted by a signal after it has
 57452 successfully read some data, it returns the number of bytes read. In Issue 3, it was optional
 57453 whether *read()* returned the number of bytes read, or whether it returned `-1` with *errno* set
 57454 to `[EINTR]`. This is a FIPS requirement.

57455 In the DESCRIPTION, text is added to indicate that for regular files, no data transfer
 57456 occurs past the offset maximum established in the open file description associated with
 57457 *files*. This change is to support large files.

57458 The `[Eoverflow]` mandatory error condition is added.

- 57459 The [ENXIO] optional error condition is added.
- 57460 Text referring to sockets is added to the DESCRIPTION.
- 57461 The following changes were made to align with the IEEE P1003.1a draft standard:
- 57462 The effect of reading zero bytes is clarified.
- 57463 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
57464 *read()* results are unspecified for typed memory objects.
- 57465 New RATIONALE is added to explain the atomicity requirements for input and output
57466 operations.
- 57467 The following error conditions are added for operations on sockets: [EAGAIN],
57468 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].
- 57469 The [EIO] error is made optional.
- 57470 The following error conditions are added for operations on sockets: [ENOBUFS] and
57471 [ENOMEM].
- 57472 The *readv()* function is split out into a separate reference page.
- 57473 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/108 is applied, updating the [EAGAIN]
57474 error in the ERRORS section from “the process would be delayed” to “the thread would be
57475 delayed”.
- 57476 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/109 is applied, making an editorial
57477 correction in the RATIONALE section.
- 57478 **Issue 7**
- 57479 The *pread()* function is moved from the XSI option to the Base.
- 57480 Functionality relating to the XSI STREAMS option is marked obsolescent.
- 57481 Changes are made related to support for finegrained timestamps.
- 57482 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0480 [218], XSH/TC1-2008/0481 [79],
57483 XSH/TC1-2008/0482 [218], XSH/TC1-2008/0483 [218], XSH/TC1-2008/0484 [218], and
57484 XSH/TC1-2008/0485 [218,428] are applied.
- 57485 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0302 [710] and XSH/TC2-2008/0303
57486 [676,710] are applied.

57487 **NAME**

57488 readdir, readdir_r — read a directory

57489 **SYNOPSIS**

57490 #include <dirent.h>

57491 struct dirent *readdir(DIR *dirp);

57492 int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,

57493 struct dirent **restrict result);

57494 **DESCRIPTION**

57495 The type **DIR**, which is defined in the **<dirent.h>** header, represents a *directory stream*, which is
 57496 an ordered sequence of all the directory entries in a particular directory. Directory entries
 57497 represent files; files may be removed from a directory or added to a directory asynchronously to
 57498 the operation of *readdir()*.

57499 The *readdir()* function shall return a pointer to a structure representing the directory entry at the
 57500 current position in the directory stream specified by the argument *dirp*, and position the
 57501 directory stream at the next entry. It shall return a null pointer upon reaching the end of the
 57502 directory stream. The structure **dirent** defined in the **<dirent.h>** header describes a directory
 57503 entry. The value of the structure's *d_ino* member shall be set to the file serial number of the file
 57504 named by the *d_name* member. If the *d_name* member names a symbolic link, the value of the
 57505 *d_ino* member shall be set to the file serial number of the symbolic link itself.

57506 The *readdir()* function shall not return directory entries containing empty names. If entries for
 57507 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-
 57508 dot; otherwise, they shall not be returned.

57509 The application shall not modify the structure to which the return value of *readdir()* points, nor
 57510 any storage areas pointed to by pointers within the structure. The returned pointer, and pointers
 57511 within the structure, might be invalidated or the structure or the storage areas might be
 57512 overwritten by a subsequent call to *readdir()* on the same directory stream. They shall not be
 57513 affected by a call to *readdir()* on a different directory stream. The returned pointer, and pointers
 57514 within the structure, might also be invalidated if the calling thread is terminated.

57515 If a file is removed from or added to the directory after the most recent call to *opendir()* or
 57516 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

57517 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()*
 57518 shall mark for update the last data access timestamp of the directory each time the directory is
 57519 actually read.

57520 After a call to *fork()*, either the parent or child (but not both) may continue processing the
 57521 XSI directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes
 57522 use these functions, the result is undefined.

57523 The *readdir()* function need not be thread-safe.

57524 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If
 57525 *errno* is set to non-zero on return, an error occurred.

57526 The *readdir_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the
 57527 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer
 57528 to this structure at the location referenced by *result*, and position the directory stream at the next
 57529 entry.

57530 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name*
 57531 members containing at least {NAME_MAX}+1 elements.

57532 Upon successful return, the pointer returned at **result* shall have the same value as the argument
57533 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

57534 The *readdir_r()* function shall not return directory entries containing empty names.

57535 If a file is removed from or added to the directory after the most recent call to *opendir()* or
57536 *rewinddir()*, whether a subsequent call to *readdir_r()* returns an entry for that file is unspecified.

57537 The *readdir_r()* function may buffer several directory entries per actual read operation;
57538 *readdir_r()* shall mark for update the last data access timestamp of the directory each time the
57539 directory is actually read.

57540 RETURN VALUE

57541 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.
57542 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate
57543 the error. When the end of the directory is encountered, a null pointer shall be returned and
57544 *errno* is not changed.

57545 If successful, the *readdir_r()* function shall return zero; otherwise, an error number shall be
57546 returned to indicate the error.

57547 ERRORS

57548 These functions shall fail if:

57549 [EOVERFLOW] One of the values in the structure to be returned cannot be represented
57550 correctly.

57551 These functions may fail if:

57552 [EBADF] The *dirp* argument does not refer to an open directory stream.

57553 [ENOENT] The current position of the directory stream is invalid.

57554 EXAMPLES

57555 The following sample program searches the current directory for each of the arguments supplied
57556 on the command line.

```
57557 #include <dirent.h>
57558 #include <errno.h>
57559 #include <stdio.h>
57560 #include <string.h>

57561 static void lookup(const char *arg)
57562 {
57563     DIR *dirp;
57564     struct dirent *dp;

57565     if ((dirp = opendir(".")) == NULL) {
57566         perror("couldn't open '.');
57567         return;
57568     }

57569     do {
57570         errno = 0;
57571         if ((dp = readdir(dirp)) != NULL) {
57572             if (strcmp(dp->d_name, arg) != 0)
57573                 continue;

57574             (void) printf("found %s\n", arg);
57575             (void) closedir(dirp);
```

```

57576             return;
57577         }
57578     } while (dp != NULL);
57579     if (errno != 0)
57580         perror("error reading directory");
57581     else
57582         (void) printf("failed to find %s\n", arg);
57583     (void) closedir(dirp);
57584     return;
57585 }
57586 int main(int argc, char *argv[])
57587 {
57588     int i;
57589     for (i = 1; i < argc; i++)
57590         lookup(argv[i]);
57591     return (0);
57592 }

```

APPLICATION USAGE

The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to examine the contents of the directory.

The *readdir_r()* function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be inferred.

Historical implementations of *readdir()* obtain multiple directory entries on a single read operation, which permits subsequent *readdir()* operations to operate from the buffered information. Any wording that required each successful *readdir()* operation to mark the directory last data access timestamp for update would disallow such historical performance-oriented implementations.

When returning a directory entry for the root of a mounted file system, some historical implementations of *readdir()* returned the file serial number of the underlying mount point, rather than of the root of the mounted file system. This behavior is considered to be a bug, since the underlying file serial number has no significance to applications.

Since *readdir()* returns NULL when it detects an error and when the end of the directory is encountered, an application that needs to tell the difference must set *errno* to zero before the call and check it if NULL is returned. Since the function must not change *errno* in the second case and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL indicates end-of-directory; otherwise, an error.

Routines to deal with this problem more directly were proposed:

```

57616 int derror (dirp)
57617 DIR *dirp;
57618
57618 void clearerr (dirp)
57619 DIR *dirp;

```

The first would indicate whether an error had occurred, and the second would clear the error

- 57621 indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()*
57622 not change *errno* when end-of-directory is encountered.
- 57623 An error or signal indicating that a directory has changed while open was considered but
57624 rejected.
- 57625 The thread-safe version of the directory reading function returns values in a user-supplied buffer
57626 instead of possibly using a static data area that may be overwritten by each call. Either the
57627 {NAME_MAX} compile-time constant or the corresponding *pathconf()* option can be used to
57628 determine the maximum sizes of returned pathnames.
- 57629 **FUTURE DIRECTIONS**
- 57630 None.
- 57631 **SEE ALSO**
- 57632 *closedir()*, *dirfd()*, *exec*, *fdopendir()*, *fstatat()*, *rewinddir()*, *symlink()*
- 57633 XBD **<dirent.h>**, **<sys/types.h>**
- 57634 **CHANGE HISTORY**
- 57635 First released in Issue 2.
- 57636 **Issue 5**
- 57637 Large File Summit extensions are added.
- 57638 The *readdir_r()* function is included for alignment with the POSIX Threads Extension.
- 57639 A note indicating that the *readdir()* function need not be reentrant is added to the
57640 DESCRIPTION.
- 57641 **Issue 6**
- 57642 The *readdir_r()* function is marked as part of the Thread-Safe Functions option.
- 57643 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.
- 57644 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful
57645 return for the *readdir_r()* function.
- 57646 The following new requirements on POSIX implementations derive from alignment with the
57647 Single UNIX Specification:
- 57648 The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
57649 required for conforming implementations of previous POSIX specifications, it was not
57650 required for UNIX applications.
- 57651 A statement is added to the DESCRIPTION indicating the disposition of certain fields in
57652 **struct dirent** when an entry refers to a symbolic link.
- 57653 The [Eoverflow] mandatory error condition is added. This change is to support large
57654 files.
- 57655 The [ENOENT] optional error condition is added.
- 57656 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
57657 its avoidance of possibly using a static data area.
- 57658 The **restrict** keyword is added to the *readdir_r()* prototype for alignment with the
57659 ISO/IEC 9899:1999 standard.
- 57660 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES
57661 section with a new example.

57662 **Issue 7**

57663 Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

57664 Austin Group Interpretation 1003.1-2001 #156 is applied.

57665 The *readdir_r()* function is moved from the Thread-Safe Functions option to the Base.

57666 Changes are made related to support for finegrained timestamps.

57667 The value of the *d_ino* member is no longer unspecified for symbolic links.

57668 SD5-XSH-ERN-193 is applied.

57669 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0486 [75] is applied.

57670 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0304 [656] is applied.

57671 **NAME**

57672 readlink, readlinkat — read the contents of a symbolic link

57673 **SYNOPSIS**

57674 #include <unistd.h>

57675 ssize_t readlink(const char *restrict path, char *restrict buf,
57676 size_t bufsize);

57677 OH #include <fcntl.h>

57678 ssize_t readlinkat(int fd, const char *restrict path,
57679 char *restrict buf, size_t bufsize);57680 **DESCRIPTION**57681 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the
57682 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,
57683 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to
57684 contain the link content, the first *bufsize* bytes shall be placed in *buf*.57685 If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.57686 Upon successful completion, *readlink()* shall mark for update the last data access timestamp of
57687 the symbolic link.57688 The *readlinkat()* function shall be equivalent to the *readlink()* function except in the case where
57689 *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the
57690 directory associated with the file descriptor *fd* instead of the current working directory. If the
57691 access mode of the open file description associated with the file descriptor is not O_SEARCH,
57692 the function shall check whether directory searches are permitted using the current permissions
57693 of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function
57694 shall not perform the check.57695 If *readlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
57696 directory shall be used and the behavior shall be identical to a call to *readlink()*.57697 **RETURN VALUE**57698 Upon successful completion, these functions shall return the count of bytes placed in the buffer.
57699 Otherwise, these functions shall return a value of -1, leave the buffer unchanged, and set *errno* to
57700 indicate the error.57701 **ERRORS**

57702 These functions shall fail if:

57703 [EACCES] Search permission is denied for a component of the path prefix of *path*.57704 [EINVAL] The *path* argument names a file that is not a symbolic link.

57705 [EIO] An I/O error occurred while reading from the file system.

57706 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
57707 argument.

57708 [ENAMETOOLONG]

57709 The length of a component of a pathname is longer than {NAME_MAX}.

57710 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.57711 [ENOTDIR] A component of the path prefix names an existing file that is neither a
57712 directory nor a symbolic link to a directory, or the *path* argument contains at
57713 least one non-*<slash>* character and ends with one or more trailing *<slash>*

57714 characters and the last pathname component names an existing file that is
57715 neither a directory nor a symbolic link to a directory.

57716 The *readlinkat()* function shall fail if:

57717 [EACCES] The access mode of the open file description associated with *fd* is not
57718 O_SEARCH and the permissions of the directory underlying *fd* do not permit
57719 directory searches.

57720 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
57721 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

57722 [ENOTDIR] The *path* argument is not an absolute path and *fd* is a file descriptor associated
57723 with a non-directory file.

57724 These functions may fail if:

57725 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
57726 resolution of the *path* argument.

57727 [ENAMETOOLONG]
57728 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
57729 symbolic link produced an intermediate result with a length that exceeds
57730 {PATH_MAX}.

57731 EXAMPLES

57732 Reading the Name of a Symbolic Link

57733 The following example shows how to read the name of a symbolic link named */modules/pass1*.

```
57734 #include <unistd.h>
57735 char buf[1024];
57736 ssize_t len;
57737 ...
57738 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
57739     buf[len] = '\0';
```

57740 APPLICATION USAGE

57741 Conforming applications should not assume that the returned contents of the symbolic link are
57742 null-terminated.

57743 RATIONALE

57744 The type associated with *bufsiz* is a **size_t** in order to be consistent with both the ISO C standard
57745 and the definition of *read()*. The behavior specified for *readlink()* when *bufsiz* is zero represents
57746 historical practice. For this case, the standard developers considered a change whereby *readlink()*
57747 would return the number of non-null bytes contained in the symbolic link with the buffer *buf*
57748 remaining unchanged; however, since the **stat** structure member *st_size* value can be used to
57749 determine the size of buffer necessary to contain the contents of the symbolic link as returned by
57750 *readlink()*, this proposal was rejected, and the historical practice retained.

57751 The purpose of the *readlinkat()* function is to read the content of symbolic links in directories
57752 other than the current working directory without exposure to race conditions. Any part of the
57753 path of a file could be changed in parallel to a call to *readlink()*, resulting in unspecified behavior.
57754 By opening a file descriptor for the target directory and using the *readlinkat()* function it can be
57755 guaranteed that the symbolic link read is located relative to the desired directory.

57756 **FUTURE DIRECTIONS**

57757 None.

57758 **SEE ALSO**57759 *fstatat()*, *symlink()*57760 XBD [<fcntl.h>](#), [<unistd.h>](#)57761 **CHANGE HISTORY**

57762 First released in Issue 4, Version 2.

57763 **Issue 5**

57764 Moved from X/OPEN UNIX extension to BASE.

57765 **Issue 6**57766 The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.57767 The following new requirements on POSIX implementations derive from alignment with the
57768 Single UNIX Specification:

57769 This function is made mandatory.

57770 In this function it is possible for the return value to exceed the range of the type **ssize_t**
57771 (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the
57772 size of the **size_t** object is added to the description to resolve this conflict.

57773 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

57774 The FUTURE DIRECTIONS section is changed to None.

57775 The following changes were made to align with the IEEE P1003.1a draft standard:

57776 The [ELOOP] optional error condition is added.

57777 The **restrict** keyword is added to the *readlink()* prototype for alignment with the
57778 ISO/IEC 9899: 1999 standard.57779 **Issue 7**

57780 Austin Group Interpretation 1003.1-2001 #143 is applied.

57781 SD5-XSH-ERN-189 is applied, updating the ERRORS section.

57782 The *readlinkat()* function is added from The Open Group Technical Standard, 2006, Extended
57783 API Set Part 2.

57784 The [EACCES] error is removed from the "may fail" error conditions.

57785 Changes are made to allow a directory to be opened for searching.

57786 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
57787 pathname exists but is not a directory or a symbolic link to a directory.57788 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0487 [120], XSH/TC1-2008/0488 [461],
57789 XSH/TC1-2008/0489 [143], XSH/TC1-2008/0490 [324], XSH/TC1-2008/0491 [278],
57790 XSH/TC1-2008/0492 [278], XSH/TC1-2008/0493 [455], and XSH/TC1-2008/0494 [151,231] are
57791 applied.57792 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0305 [591], XSH/TC2-2008/0306 [817],
57793 XSH/TC2-2008/0307 [817], and XSH/TC2-2008/0308 [591] are applied.

57794 **NAME**

57795 readv — read a vector

57796 **SYNOPSIS**

```
57797 XSI #include <sys/uio.h>
57798 ssize_t readv(int fd, const struct iovec *iov, int iovcnt);
```

57799 **DESCRIPTION**

57800 The *readv()* function shall be equivalent to *read()*, except as described below. The *readv()*
 57801 function shall place the input data into the *iovcnt* buffers specified by the members of the *iov*
 57802 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than
 57803 or equal to {IOV_MAX}.

57804 Each *iovec* entry specifies the base address and length of an area in memory where data should
 57805 be placed. The *readv()* function shall always fill an area completely before proceeding to the
 57806 next.

57807 Upon successful completion, *readv()* shall mark for update the last data access timestamp of the
 57808 file.

57809 **RETURN VALUE**57810 Refer to *read()*.57811 **ERRORS**57812 Refer to *read()*.57813 In addition, the *readv()* function shall fail if:57814 [EINVAL] The sum of the *iov_len* values in the *iov* array overflowed an *ssize_t*.57815 The *readv()* function may fail if:57816 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.57817 **EXAMPLES**57818 **Reading Data into an Array**

57819 The following example reads data from the file associated with the file descriptor *fd* into the
 57820 buffers specified by members of the *iov* array.

```
57821 #include <sys/types.h>
57822 #include <sys/uio.h>
57823 #include <unistd.h>
57824 ...
57825 ssize_t bytes_read;
57826 int fd;
57827 char buf0[20];
57828 char buf1[30];
57829 char buf2[40];
57830 int iovcnt;
57831 struct iovec iov[3];

57832 iov[0].iov_base = buf0;
57833 iov[0].iov_len = sizeof(buf0);
57834 iov[1].iov_base = buf1;
57835 iov[1].iov_len = sizeof(buf1);
57836 iov[2].iov_base = buf2;
```

```
57837     iov[2].iov_len = sizeof(buf2);
57838     ...
57839     iovcnt = sizeof(iov) / sizeof(struct iovec);
57840     bytes_read = readv(fd, iov, iovcnt);
57841     ...
```

57842 APPLICATION USAGE

57843 None.

57844 RATIONALE

57845 Refer to *read()*.

57846 FUTURE DIRECTIONS

57847 None.

57848 SEE ALSO

57849 *read()*, *writev()*

57850 XBD <sys/uio.h>

57851 CHANGE HISTORY

57852 First released in Issue 4, Version 2.

57853 Issue 6

57854 Split out from the *read()* reference page.

57855 Issue 7

57856 Changes are made related to support for finegrained timestamps.

57857 **NAME**

57858 realloc — memory reallocator

57859 **SYNOPSIS**

57860 #include <stdlib.h>

57861 void *realloc(void *ptr, size_t size);

57862 **DESCRIPTION**

57863 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 57864 conflict between the requirements described here and the ISO C standard is unintentional. This
 57865 volume of POSIX.1-2017 defers to the ISO C standard.

57866 The *realloc()* function shall deallocate the old object pointed to by *ptr* and return a pointer to a
 57867 new object that has the size specified by *size*. The contents of the new object shall be the same as
 57868 that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in
 57869 the new object beyond the size of the old object have indeterminate values. If the size of the
 57870 space requested is zero, the behavior shall be implementation-defined: either a null pointer is
 57871 returned, or the behavior shall be as if the size were some non-zero value, except that the
 57872 behavior is undefined if the returned pointer is used to access an object. If the space cannot be
 57873 allocated, the object shall remain unchanged.

57874 If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

57875 If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has
 57876 previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

57877 The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The
 57878 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 57879 a pointer to any type of object and then used to access such an object in the space allocated (until
 57880 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 57881 disjoint from any other object. The pointer returned shall point to the start (lowest byte address)
 57882 of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

57883 **RETURN VALUE**

57884 Upon successful completion, *realloc()* shall return a pointer to the (possibly moved) allocated
 57885 space. If *size* is 0, either:

57886 CX A null pointer shall be returned and, if *ptr* is not a null pointer, *errno* shall be set to an
 57887 implementation-defined value.

57888 A pointer to the allocated space shall be returned, and the memory object pointed to by *ptr*
 57889 shall be freed. The application shall ensure that the pointer is not used to access an object.

57890 CX If there is not enough available memory, *realloc()* shall return a null pointer and set *errno* to
 57891 CX [ENOMEM]. If *realloc()* returns a null pointer and *errno* has been set to [ENOMEM], the
 57892 memory referenced by *ptr* shall not be changed.

57893 **ERRORS**

57894 The *realloc()* function shall fail if:

57895 CX [ENOMEM] Insufficient memory is available.

57896 EXAMPLES

57897 None.

57898 APPLICATION USAGE

57899 The description of `realloc()` has been modified from previous versions of this standard to align
57900 with the ISO/IEC 9899:1999 standard. Previous versions explicitly permitted a call to `realloc(p, 0)`
57901 to free the space pointed to by `p` and return a null pointer. While this behavior could be interpreted as
57902 permitted by this version of the standard, the C language committee have indicated that this interpretation
57903 is incorrect. Applications should assume that if `realloc()` returns a null pointer, the space pointed to by `p`
57904 has not been freed. Since this could lead to double-frees, implementations should also set `errno` if a null
57905 pointer actually indicates a failure, and applications should only free the space if `errno` was changed.

57906 RATIONALE

57907 None.

57908 FUTURE DIRECTIONS

57909 This standard defers to the ISO C standard. While that standard currently has language that
57910 might permit `realloc(p, 0)`, where `p` is not a null pointer, to free `p` while still returning a null pointer, the
57911 committee responsible for that standard is considering clarifying the language to explicitly prohibit that
57912 alternative.

57913 SEE ALSO

57914 [`calloc\(\)`](#), [`free\(\)`](#), [`malloc\(\)`](#)

57915 XBD [`<stdlib.h>`](#)

57916 CHANGE HISTORY

57917 First released in Issue 1. Derived from Issue 1 of the SVID.

57918 Issue 6

57919 Extensions beyond the ISO C standard are marked.

57920 The following new requirements on POSIX implementations derive from alignment with the
57921 Single UNIX Specification:

57922 In the RETURN VALUE section, if there is not enough available memory, the setting of
57923 `errno` to [ENOMEM] is added.

57924 The [ENOMEM] error condition is added.

57925 Issue 7

57926 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0495 [400], XSH/TC1-2008/0496 [400],
57927 XSH/TC1-2008/0497 [400], and XSH/TC1-2008/0498 [400] are applied.

57928 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0309 [526] and XSH/TC2-2008/0310
57929 [526,688] are applied.

57930 **NAME**

57931 realpath — resolve a pathname

57932 **SYNOPSIS**

```
57933 XSI      #include <stdlib.h>
57934         char *realpath(const char *restrict file_name,
57935                        char *restrict resolved_name);
```

57936 **DESCRIPTION**

57937 The *realpath()* function shall derive, from the pathname pointed to by *file_name*, an absolute
57938 pathname that resolves to the same directory entry, whose resolution does not involve '.',
57939 '..', or symbolic links. If *resolved_name* is a null pointer, the generated pathname shall be
57940 stored as a null-terminated string in a buffer allocated as if by a call to *malloc()*. Otherwise, if
57941 {PATH_MAX} is defined as a constant in the <limits.h> header, then the generated pathname
57942 shall be stored as a null-terminated string, up to a maximum of {PATH_MAX} bytes, in the
57943 buffer pointed to by *resolved_name*.

57944 If *resolved_name* is not a null pointer and {PATH_MAX} is not defined as a constant in the
57945 <limits.h> header, the behavior is undefined.

57946 **RETURN VALUE**

57947 Upon successful completion, *realpath()* shall return a pointer to the buffer containing the
57948 resolved name. Otherwise, *realpath()* shall return a null pointer and set *errno* to indicate the
57949 error.

57950 If the *resolved_name* argument is a null pointer, the pointer returned by *realpath()* can be passed
57951 to *free()*.

57952 If the *resolved_name* argument is not a null pointer and the *realpath()* function fails, the contents
57953 of the buffer pointed to by *resolved_name* are undefined.

57954 **ERRORS**

57955 The *realpath()* function shall fail if:

57956 [EACCES] Search permission was denied for a component of the path prefix of *file_name*.

57957 [EINVAL] The *file_name* argument is a null pointer.

57958 [EIO] An error occurred while reading from the file system.

57959 [ELOOP] A loop exists in symbolic links encountered during resolution of the *file_name*
57960 argument.

57961 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.

57963 [ENOENT] A component of *file_name* does not name an existing file or *file_name* points to
57964 an empty string.

57965 [ENOTDIR] A component of the path prefix names an existing file that is neither a
57966 directory nor a symbolic link to a directory, or the *file_name* argument contains
57967 at least one non-*<slash>* character and ends with one or more trailing *<slash>*
57968 characters and the last pathname component names an existing file that is
57969 neither a directory nor a symbolic link to a directory.

- 57970 The *realpath()* function may fail if:
- 57971 [EACCES] The *file_name* argument does not begin with a <slash> and none of the
57972 symbolic links (if any) processed during pathname resolution of *file_name* had
57973 contents that began with a <slash>, and either search permission was denied
57974 for the current directory or read or search permission was denied for a
57975 directory above the current directory in the file hierarchy.
- 57976 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
57977 resolution of the *file_name* argument.
- 57978 [ENAMETOOLONG]
57979 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
57980 symbolic link produced an intermediate result with a length that exceeds
57981 {PATH_MAX}.
- 57982 [ENOMEM] Insufficient storage space is available.

57983 EXAMPLES

57984 Generating an Absolute Pathname

57985 The following example generates an absolute pathname for the file identified by the *symlinkpath*
57986 argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
57987 #include <stdlib.h>
57988 ...
57989 char *symlinkpath = "/tmp/symlink/file";
57990 char *actualpath;
57991
57992 actualpath = realpath(symlinkpath, NULL);
57993 if (actualpath != NULL)
57994 {
57995     ... use actualpath ...
57996     free(actualpath);
57997 }
57998 else
57999 {
58000     ... handle error ...
58001 }
```

58001 APPLICATION USAGE

58002 For functions that allocate memory as if by *malloc()*, the application should release such memory
58003 when it is no longer required by a call to *free()*. For *realpath()*, this is the return value.

58004 RATIONALE

58005 Since *realpath()* has no *length* argument, if {PATH_MAX} is not defined as a constant in
58006 <limits.h>, applications have no way of determining how large a buffer they need to allocate for
58007 it to be safe to pass to *realpath()*. A {PATH_MAX} value obtained from a prior *pathconf()* call is
58008 out-of-date by the time *realpath()* is called. Hence the only reliable way to use *realpath()* when
58009 {PATH_MAX} is not defined in <limits.h> is to pass a null pointer for *resolved_name* so that
58010 *realpath()* will allocate a buffer of the necessary size.

58011 **FUTURE DIRECTIONS**

58012 None.

58013 **SEE ALSO**58014 *fpathconf()*, *free()*, *getcwd()*, *sysconf()*

58015 XBD <limits.h>, <stdlib.h>

58016 **CHANGE HISTORY**

58017 First released in Issue 4, Version 2.

58018 **Issue 5**

58019 Moved from X/OPEN UNIX extension to BASE.

58020 **Issue 6**58021 The **restrict** keyword is added to the *realpath()* prototype for alignment with the
58022 ISO/IEC 9899:1999 standard.58023 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
58024 [ELOOP] error condition is added.58025 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the
58026 DESCRIPTION for the case when *resolved_name* is a null pointer, changing the [EINVAL] error
58027 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.58028 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/110 is applied, updating the ERRORS
58029 section to refer to the *file_name* argument, rather than a nonexistent *path* argument.58030 **Issue 7**

58031 Austin Group Interpretation 1003.1-2001 #143 is applied.

58032 This function is updated for passing a null pointer to *realpath()* for the *resolved_name* argument.58033 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
58034 *malloc()*.58035 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0499 [353], XSH/TC1-2008/0500 [324],
58036 and XSH/TC1-2008/0501 [353] are applied.

58037 **NAME**

58038 recv — receive a message from a connected socket

58039 **SYNOPSIS**

58040 #include <sys/socket.h>

58041 ssize_t recv(int *socket*, void **buffer*, size_t *length*, int *flags*);58042 **DESCRIPTION**

58043 The *recv()* function shall receive a message from a connection-mode or connectionless-mode
 58044 socket. It is normally used with connected sockets because it does not permit the application to
 58045 retrieve the source address of received data.

58046 The *recv()* function takes the following arguments:

58047 *socket* Specifies the socket file descriptor.

58048 *buffer* Points to a buffer where the message should be stored.

58049 *length* Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

58050 *flags* Specifies the type of message reception. Values of this argument are formed by
 58051 logically OR'ing zero or more of the following values:

58052 MSG_PEEK Peeks at an incoming message. The data is treated as unread and
 58053 the next *recv()* or similar function shall still return this data.

58054 MSG_OOB Requests out-of-band data. The significance and semantics of
 58055 out-of-band data are protocol-specific.

58056 MSG_WAITALL On SOCK_STREAM sockets this requests that the function block
 58057 until the full amount of data can be returned. The function may
 58058 return the smaller amount of data if the socket is a message-
 58059 based socket, if a signal is caught, if the connection is terminated,
 58060 if MSG_PEEK was specified, or if an error is pending for the
 58061 socket.

58062 The *recv()* function shall return the length of the message written to the buffer pointed to by the
 58063 *buffer* argument. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET,
 58064 the entire message shall be read in a single operation. If a message is too long to fit in the
 58065 supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be
 58066 discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be
 58067 ignored. In this case, data shall be returned to the user as soon as it becomes available, and no
 58068 data shall be discarded.

58069 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 58070 message.

58071 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 58072 descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket
 58073 and O_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to
 58074 [EAGAIN] or [EWOULDBLOCK].

58075 **RETURN VALUE**

58076 Upon successful completion, *recv()* shall return the length of the message in bytes. If no
 58077 messages are available to be received and the peer has performed an orderly shutdown, *recv()*
 58078 shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

58079 **ERRORS**58080 The *recv()* function shall fail if:

58081 [EAGAIN] or [EWOULDBLOCK]

58082 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
58083 to be received; or MSG_OOB is set and no out-of-band data is available and
58084 either the socket's file descriptor is marked O_NONBLOCK or the socket does
58085 not support blocking to await out-of-band data.58086 [EBADF] The *socket* argument is not a valid file descriptor.

58087 [ECONNRESET] A connection was forcibly closed by a peer.

58088 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any
58089 data was available.

58090 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

58091 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

58092 [ENOTSOCK] The *socket* argument does not refer to a socket.

58093 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.

58094 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
58095 transmission timeout on active connection.58096 The *recv()* function may fail if:

58097 [EIO] An I/O error occurred while reading from or writing to the file system.

58098 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

58099 [ENOMEM] Insufficient memory was available to fulfill the request.

58100 **EXAMPLES**

58101 None.

58102 **APPLICATION USAGE**58103 The *recv()* function is equivalent to *recvfrom()* with null pointer *address* and *address_len*
58104 arguments, and to *read()* if the *socket* argument refers to a socket and the *flags* argument is 0.58105 The *select()* and *poll()* functions can be used to determine when data is available to be received.58106 **RATIONALE**

58107 None.

58108 **FUTURE DIRECTIONS**

58109 None.

58110 **SEE ALSO**58111 *poll()*, *pselect()*, *read()*, *recvmsg()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*,
58112 *write()*

58113 XBD <sys/socket.h>

58114 **CHANGE HISTORY**

58115 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

58116 **Issue 7**
58117

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0502 [462] is applied.

58118 **NAME**

58119 recvfrom — receive a message from a socket

58120 **SYNOPSIS**

```
58121 #include <sys/socket.h>
58122 ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
58123                 int flags, struct sockaddr *restrict address,
58124                 socklen_t *restrict address_len);
```

58125 **DESCRIPTION**

58126 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode
 58127 socket. It is normally used with connectionless-mode sockets because it permits the application
 58128 to retrieve the source address of received data.

58129 The *recvfrom()* function takes the following arguments:

58130	<i>socket</i>	Specifies the socket file descriptor.
58131	<i>buffer</i>	Points to the buffer where the message should be stored.
58132	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
58133	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by 58134 logically OR'ing zero or more of the following values:
58135	MSG_PEEK	Peeks at an incoming message. The data is treated as unread 58136 and the next <i>recvfrom()</i> or similar function shall still return 58137 this data.
58138	MSG_OOB	Requests out-of-band data. The significance and semantics 58139 of out-of-band data are protocol-specific.
58140	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function 58141 block until the full amount of data can be returned. The 58142 function may return the smaller amount of data if the socket 58143 is a message-based socket, if a signal is caught, if the 58144 connection is terminated, if MSG_PEEK was specified, or if 58145 an error is pending for the socket.
58146	<i>address</i>	A null pointer, or points to a sockaddr structure in which the sending address 58147 is to be stored. The length and format of the address depend on the address 58148 family of the socket.
58149	<i>address_len</i>	Either a null pointer, if <i>address</i> is a null pointer, or a pointer to a socklen_t 58150 object which on input specifies the length of the supplied sockaddr structure, 58151 and on output specifies the length of the stored address.

58152 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by
 58153 RS the *buffer* argument. For message-based sockets, such as **SOCK_RAW**, **SOCK_DGRAM**, and
 58154 **SOCK_SEQPACKET**, the entire message shall be read in a single operation. If a message is too
 58155 long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess
 58156 bytes shall be discarded. For stream-based sockets, such as **SOCK_STREAM**, message
 58157 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes
 58158 available, and no data shall be discarded.

58159 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 58160 message.

58161 Not all protocols provide the source address for messages. If the *address* argument is not a null

58162 pointer and the protocol provides the source address of messages, the source address of the
 58163 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,
 58164 and the length of this address shall be stored in the object pointed to by the *address_len*
 58165 argument.

58166 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 58167 the stored address shall be truncated.

58168 If the *address* argument is not a null pointer and the protocol does not provide the source address
 58169 of messages, the value stored in the object pointed to by *address* is unspecified.

58170 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 58171 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the
 58172 socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno*
 58173 to [EAGAIN] or [EWOULDBLOCK].

58174 RETURN VALUE

58175 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no
 58176 messages are available to be received and the peer has performed an orderly shutdown,
 58177 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the
 58178 error.

58179 ERRORS

58180 The *recvfrom()* function shall fail if:

58181 [EAGAIN] or [EWOULDBLOCK]

58182 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
 58183 to be received; or MSG_OOB is set and no out-of-band data is available and
 58184 either the socket's file descriptor is marked O_NONBLOCK or the socket does
 58185 not support blocking to await out-of-band data.

58186 [EBADF] The *socket* argument is not a valid file descriptor.

58187 [ECONNRESET] A connection was forcibly closed by a peer.

58188 [EINTR] A signal interrupted *recvfrom()* before any data was available.

58189 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

58190 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

58191 [ENOTSOCK] The *socket* argument does not refer to a socket.

58192 [EOPNOTSUPP] The specified flags are not supported for this socket type.

58193 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 58194 transmission timeout on active connection.

58195 The *recvfrom()* function may fail if:

58196 [EIO] An I/O error occurred while reading from or writing to the file system.

58197 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

58198 [ENOMEM] Insufficient memory was available to fulfill the request.

58199 **EXAMPLES**

58200 None.

58201 **APPLICATION USAGE**58202 The *select()* and *poll()* functions can be used to determine when data is available to be received.58203 **RATIONALE**

58204 None.

58205 **FUTURE DIRECTIONS**

58206 None.

58207 **SEE ALSO**58208 *poll()*, *pselect()*, *read()*, *recv()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write()*58209 XBD <[sys/socket.h](#)>58210 **CHANGE HISTORY**

58211 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

58212 **Issue 7**

58213 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0503 [464] is applied.

58214 **NAME**

58215 recvmsg — receive a message from a socket

58216 **SYNOPSIS**

58217 #include <sys/socket.h>

58218 ssize_t recvmsg(int socket, struct msghdr *message, int flags);

58219 **DESCRIPTION**

58220 The *recvmsg()* function shall receive a message from a connection-mode or connectionless-mode
 58221 socket. It is normally used with connectionless-mode sockets because it permits the application
 58222 to retrieve the source address of received data.

58223 The *recvmsg()* function takes the following arguments:

58224	<i>socket</i>	Specifies the socket file descriptor.
58225	<i>message</i>	Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.
58226		
58227		
58228		
58229	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
58230		
58231	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
58232		
58233	MSG_PEEK	Peeks at the incoming message.
58234	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.
58235		
58236		
58237		
58238		
58239		

58240 The *recvmsg()* function shall receive messages from unconnected or connected sockets and shall
 58241 return the length of the message.

58242 The *recvmsg()* function shall return the total length of the message. For message-based sockets,
 58243 such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single
 58244 operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the
 58245 *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the
 58246 *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM,
 58247 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it
 58248 becomes available, and no data shall be discarded.

58249 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 58250 message.

58251 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 58252 descriptor, *recvmsg()* shall block until a message arrives. If no messages are available at the
 58253 socket and O_NONBLOCK is set on the socket's file descriptor, the *recvmsg()* function shall fail
 58254 and set *errno* to [EAGAIN] or [EWOULDBLOCK].

58255 In the **msghdr** structure, the *msg_name* member may be a null pointer if the source address is not
 58256 required. Otherwise, if the socket is unconnected, the *msg_name* member points to a **sockaddr**
 58257 structure in which the source address is to be stored, and the *msg_namelen* member on input
 58258 specifies the length of the supplied **sockaddr** structure and on output specifies the length of the

58259 stored address. If the actual length of the address is greater than the length of the supplied
 58260 **sockaddr** structure, the stored address shall be truncated. If the socket is connected, the
 58261 *msg_name* and *msg_namelen* members shall be ignored. The *msg_iov* and *msg_iovlen* fields are
 58262 used to specify where the received data shall be stored. The *msg_iov* member points to an array
 58263 of **iovec** structures; the *msg_iovlen* member shall be set to the dimension of this array. In each
 58264 **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in
 58265 bytes. Each storage area indicated by *msg_iov* is filled with received data in turn until all of the
 58266 received data is stored or all of the areas have been filled.

58267 Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-
 58268 inclusive OR of all of the following flags that indicate conditions detected for the received
 58269 message:

58270 MSG_EOR End-of-record was received (if supported by the protocol).

58271 MSG_OOB Out-of-band data was received.

58272 MSG_TRUNC Normal data was truncated.

58273 MSG_CTRUNC Control data was truncated.

58274 RETURN VALUE

58275 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no
 58276 messages are available to be received and the peer has performed an orderly shutdown,
 58277 *recvmsg()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

58278 ERRORS

58279 The *recvmsg()* function shall fail if:

58280 [EAGAIN] or [EWOULDBLOCK]

58281 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
 58282 to be received; or MSG_OOB is set and no out-of-band data is available and
 58283 either the socket's file descriptor is marked O_NONBLOCK or the socket does
 58284 not support blocking to await out-of-band data.

58285 [EBADF] The *socket* argument is not a valid open file descriptor.

58286 [ECONNRESET] A connection was forcibly closed by a peer.

58287 [EINTR] This function was interrupted by a signal before any data was available.

58288 [EINVAL] The sum of the *iov_len* values overflows a **ssize_t**, or the MSG_OOB flag is set
 58289 and no out-of-band data is available.

58290 [EMSGSIZE] The *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less
 58291 than or equal to 0, or is greater than {IOV_MAX}.

58292 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

58293 [ENOTSOCK] The *socket* argument does not refer to a socket.

58294 [EOPNOTSUPP] The specified flags are not supported for this socket type.

58295 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 58296 transmission timeout on active connection.

58297 The *recvmsg()* function may fail if:

58298 [EIO] An I/O error occurred while reading from or writing to the file system.

58299 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

58300 [ENOMEM] Insufficient memory was available to fulfill the request.

58301 **EXAMPLES**

58302 None.

58303 **APPLICATION USAGE**

58304 The *select()* and *poll()* functions can be used to determine when data is available to be received.

58305 **RATIONALE**

58306 None.

58307 **FUTURE DIRECTIONS**

58308 None.

58309 **SEE ALSO**

58310 *poll()*, *pselect()*, *recv()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*

58311 XBD <[sys/socket.h](#)>

58312 **CHANGE HISTORY**

58313 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

58314 **Issue 7**

58315 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0504 [464] is applied.

58316 **NAME**
 58317 regcomp, regerror, regex, regfree — regular expression matching

58318 **SYNOPSIS**
 58319 #include <regex.h>
 58320 int regcomp(regex_t *restrict preg, const char *restrict pattern,
 58321 int cflags);
 58322 size_t regerror(int errcode, const regex_t *restrict preg,
 58323 char *restrict errbuf, size_t errbuf_size);
 58324 int regex(const regex_t *restrict preg, const char *restrict string,
 58325 size_t nmatch, regmatch_t pmatch[restrict], int eflags);
 58326 void regfree(regex_t *preg);

58327 **DESCRIPTION**
 58328 These functions interpret *basic* and *extended* regular expressions as described in XBD [Chapter 9](#)
 58329 (on page 181).

58330 The **regex_t** structure is defined in **<regex.h>** and contains at least the following member:

Member Type	Member Name	Description
size_t	re_nsub	Number of parenthesized subexpressions.

58334 The **regmatch_t** structure is defined in **<regex.h>** and contains at least the following members:

Member Type	Member Name	Description
regoff_t	rm_so	Byte offset from start of <i>string</i> to start of substring.
regoff_t	rm_eo	Byte offset from start of <i>string</i> of the first character after the end of substring.

58340 The *regcomp()* function shall compile the regular expression contained in the string pointed to by
 58341 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*
 58342 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in
 58343 the **<regex.h>** header:

- 58344 REG_EXTENDED Use Extended Regular Expressions.
- 58345 REG_ICASE Ignore case in match (see XBD [Chapter 9](#), on page 181).
- 58346 REG_NOSUB Report only success/fail in *regex()*.
- 58347 REG_NEWLINE Change the handling of <newline> characters, as described in the text.

58348 The default regular expression type for *pattern* is a Basic Regular Expression. The application can
 58349 specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

58350 If the REG_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re_nsub* to the number of
 58351 parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "()" in
 58352 extended regular expressions) found in *pattern*.

58353 The *regex()* function compares the null-terminated string specified by *string* with the compiled
 58354 regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regex()*
 58355 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The
 58356 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are
 58357 defined in the **<regex.h>** header:

58358 REG_NOTBOL The first character of the string pointed to by *string* is not the beginning of
 58359 the line. Therefore, the <circumflex> character ('^'), when taken as a
 58360 special character, shall not match the beginning of *string*.

58361 REG_NOTEOL The last character of the string pointed to by *string* is not the end of the
 58362 line. Therefore, the <dollar-sign> ('\$'), when taken as a special character,
 58363 shall not match the end of *string*.

58364 If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall
 58365 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument
 58366 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that
 58367 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions
 58368 of *pattern*: *pmatch[i].rm_so* shall be the byte offset of the beginning and *pmatch[i].rm_eo* shall be
 58369 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th
 58370 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that
 58371 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*
 58372 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself
 58373 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first
 58374 *nmatch* substrings.

58375 When matching a basic or extended regular expression, any given parenthesized subexpression of
 58376 *pattern* might participate in the match of several different substrings of *string*, or it might not
 58377 match any substring even though the pattern as a whole did match. The following rules shall be
 58378 used to determine which substrings to report in *pmatch* when matching regular expressions:

58379 1. If subexpression *i* in a regular expression is not contained within another subexpression,
 58380 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall
 58381 delimit the last such match.

58382 2. If subexpression *i* is not contained within another subexpression, and it did not
 58383 participate in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A
 58384 subexpression does not participate in the match when:

58385 '*' or "\{\}" appears immediately after the subexpression in a basic regular
 58386 expression, or '*', '?', or "{ }" appears immediately after the subexpression in
 58387 an extended regular expression, and the subexpression did not match (matched 0
 58388 times)

58389 or:

58390 '| ' is used in an extended regular expression to select this subexpression or
 58391 another, and the other subexpression matched.

58392 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained
 58393 within any other subexpression that is contained within *j*, and a match of subexpression *j*
 58394 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in
 58395 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in
 58396 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start
 58397 of *string*.

58398 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1,
 58399 then the pointers in *pmatch[i]* shall also be -1.

58400 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be
 58401 the byte offset of the character or null terminator immediately following the zero-length
 58402 string.

58403 If, when *regexexec()* is called, the locale is different from when the regular expression was
 58404 compiled, the result is undefined.

58405 If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an
 58406 ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary
 58407 character except as follows:

- 58408 1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or
 58409 by any form of a non-matching list (see XBD Chapter 9, on page 181).
- 58410 2. A <circumflex> ('^ ') in *pattern*, when used to specify expression anchoring (see XBD
 58411 Section 9.3.8, on page 188), shall match the zero-length string immediately after a
 58412 <newline> in *string*, regardless of the setting of REG_NOTBOL.
- 58413 3. A <dollar-sign> ('\$ ') in *pattern*, when used to specify expression anchoring, shall match
 58414 the zero-length string immediately before a <newline> in *string*, regardless of the setting
 58415 of REG_NOTEOL.

58416 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

58417 The following constants are defined as the minimum set of error return values, although other
 58418 errors listed as implementation extensions in <regex.h> are possible:

58419	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than
58420		two numbers, first larger than second.
58421	REG_BADPAT	Invalid regular expression.
58422	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.
58423	REG_EBRACE	"\{\}" imbalance.
58424	REG_EBRACK	"[]" imbalance.
58425	REG_ECOLLATE	Invalid collating element referenced.
58426	REG_ECTYPE	Invalid character class type referenced.
58427	REG_EESCAPE	Trailing <backslash> character in pattern.
58428	REG_EPAREN	"\(\)" or "()" imbalance.
58429	REG_ERANGE	Invalid endpoint in range expression.
58430	REG_ESPACE	Out of memory.
58431	REG_ESUBREG	Number in "\digit" invalid or in error.
58432	REG_NOMATCH	<i>regexexec()</i> failed to match.

58433 If more than one error occurs in processing a function call, any one of the possible constants may
 58434 be returned, as the order of detection is unspecified.

58435 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and
 58436 *regexexec()* to unspecified printable strings. It generates a string corresponding to the value of the
 58437 *errcode* argument, which the application shall ensure is the last non-zero value returned by
 58438 *regcomp()* or *regexexec()* with the given value of *preg*. If *errcode* is not such a value, the content of
 58439 the generated string is unspecified.

58440 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexexec()* or *regcomp()*,
 58441 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not
 58442 be as detailed under some implementations.

58443 If the *errbuf_size* argument is not 0, *regerror()* shall place the generated string into the buffer of
 58444 size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit
 58445 in the buffer, *regerror()* shall truncate the string and null-terminate the result.

58446 If *errbuf_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer
 58447 needed to hold the generated string.

58448 If the *preg* argument to *regexec()* or *regfree()* is not a compiled regular expression returned by
 58449 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression
 58450 after it is given to *regfree()*.

58451 RETURN VALUE

58452 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an
 58453 integer value indicating an error as described in <**regex.h**>, and the content of *preg* is undefined.
 58454 If a code is returned, the interpretation shall be as given in <**regex.h**>.

58455 If *regcomp()* detects an invalid RE, it may return REG_BADPAT, or it may return one of the error
 58456 codes that more precisely describes the error.

58457 Upon successful completion, the *regexec()* function shall return 0. Otherwise, it shall return
 58458 REG_NOMATCH to indicate no match.

58459 Upon successful completion, the *regerror()* function shall return the number of bytes needed to
 58460 hold the entire generated string, including the null termination. If the return value is greater
 58461 than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

58462 The *regfree()* function shall not return a value.

58463 ERRORS

58464 No errors are defined.

58465 EXAMPLES

```
58466 #include <regex.h>
58467 /*
58468  * Match string against the extended regular expression in
58469  * pattern, treating errors as no match.
58470  *
58471  * Return 1 for match, 0 for no match.
58472  */
58473 int
58474 match(const char *string, char *pattern)
58475 {
58476     int    status;
58477     regex_t re;
58478     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
58479         return(0);        /* Report error. */
58480     }
58481     status = regexec(&re, string, (size_t) 0, NULL, 0);
58482     regfree(&re);
58483     if (status != 0) {
58484         return(0);        /* Report error. */
58485     }
58486     return(1);
58487 }
```

58488 The following demonstrates how the REG_NOTBOL flag could be used with *regexec()* to find all
 58489 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very
 58490 little error checking is done.)

```
58491 (void) regcomp (&re, pattern, 0);
58492 /* This call to regexec() finds the first match on the line. */
58493 error = regexec (&re, &buffer[0], 1, &pm, 0);
58494 while (error == 0) { /* While matches found. */
58495     /* Substring found between pm.rm_so and pm.rm_eo. */
58496     /* This call to regexec() finds the next match. */
58497     error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
58498 }
```

58499 APPLICATION USAGE

58500 An application could use:

```
58501 regerror(code, preg, (char *)NULL, (size_t)0)
```

58502 to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the
 58503 string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed,
 58504 static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger
 58505 buffer if it finds that this is too small.

58506 To match a pattern as described in XCU [Section 2.13](#) (on page 2382), use the *fnmatch()* function.

58507 RATIONALE

58508 The *regexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are
 58509 supplied by the application, even if some elements of *pmatch* do not correspond to
 58510 subexpressions in *pattern*. The application developer should note that there is probably no
 58511 reason for using a value of *nmatch* that is larger than *preg->re_nsub+1*.

58512 The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like
 58513 text editors. In such applications, the user supplies an RE asking the application to find a line
 58514 that matches the given expression. An anchor in such an RE anchors at the beginning or end of
 58515 any line. Such an application can pass a sequence of <newline>-separated lines to *regexec()* as a
 58516 single long string and specify REG_NEWLINE to *regcomp()* to get the desired behavior. The
 58517 application must ensure that there are no explicit <newline> characters in *pattern* if it wants to
 58518 ensure that any match occurs entirely within a single line.

58519 The REG_NEWLINE flag affects the behavior of *regexec()*, but it is in the *cflags* parameter to
 58520 *regcomp()* to allow flexibility of implementation. Some implementations will want to generate
 58521 the same compiled RE in *regcomp()* regardless of the setting of REG_NEWLINE and have
 58522 *regexec()* handle anchors differently based on the setting of the flag. Other implementations will
 58523 generate different compiled REs based on the REG_NEWLINE.

58524 The REG_ICASE flag supports the operations taken by the *grep -i* option and the historical
 58525 implementations of *ex* and *vi*. Including this flag will make it easier for application code to be
 58526 written that does the same thing as these utilities.

58527 The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather
 58528 than pointers. This allows type-safe access to both constant and non-constant strings.

58529 The type **regoff_t** is used for the elements of *pmatch[]* to ensure that the application can
 58530 represent large arrays in memory (important for an application conforming to the Shell and
 58531 Utilities volume of POSIX.1-2017).

58532 The 1992 edition of this standard required **regoff_t** to be at least as wide as **off_t**, to facilitate
 58533 future extensions in which the string to be searched is taken from a file. However, these future

58534 extensions have not appeared. The requirement rules out popular implementations with 32-bit
58535 **regoff_t** and 64-bit **off_t**, so it has been removed.

58536 The standard developers rejected the inclusion of a *regsub()* function that would be used to do
58537 substitutions for a matched RE. While such a routine would be useful to some applications, its
58538 utility would be much more limited than the matching function described here. Both RE parsing
58539 and substitution are possible to implement without support other than that required by the
58540 ISO C standard, but matching is much more complex than substituting. The only difficult part of
58541 substitution, given the information supplied by *regexexec()*, is finding the next character in a string
58542 when there can be multi-byte characters. That is a much larger issue, and one that needs a more
58543 general solution.

58544 The *errno* variable has not been used for error returns to avoid filling the *errno* name space for
58545 this feature.

58546 The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate
58547 **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used
58548 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple
58549 threads of lightweight processes. (The *preg* argument to *regexexec()* is declared with type **const**, so
58550 the implementation is not permitted to use the structure to store intermediate results.) It also
58551 allows an application to request an arbitrary number of substrings from an RE. The number of
58552 subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexexec()*,
58553 consideration was given to dropping the REG_NOSUB flag since the user can now specify this
58554 with a zero *nmatch* argument to *regexexec()*. However, keeping REG_NOSUB allows an
58555 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()* that
58556 no subexpressions need be reported. The implementation is only required to fill in *pmatch* if
58557 *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in
58558 the ISO C standard, is unsigned, so the description of *regexexec()* does not need to address
58559 negative values of *nmatch*.

58560 REG_NOTBOL was added to allow an application to do repeated searches for the same pattern
58561 in a line. If the pattern contains a <circumflex> character that should match the beginning of a
58562 line, then the pattern should only match when matched against the beginning of the line.
58563 Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent
58564 matches, but in the general case this would require parsing the expression. The need for
58565 REG_NOTEOL is not as clear; it was added for symmetry.

58566 The addition of the *regerror()* function addresses the historical need for conforming application
58567 programs to have access to error information more than “Function failed to compile/match your
58568 RE for unknown reasons”.

58569 This interface provides for two different methods of dealing with error conditions. The specific
58570 error codes (REG_EBRACE, for example), defined in <**regex.h**>, allow an application to recover
58571 from an error if it is so able. Many applications, especially those that use patterns supplied by a
58572 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-
58573 readable error message to present to the user.

58574 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating
58575 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was
58576 considered unacceptable since it creates difficulties for multi-threaded applications.

58577 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more
58578 descriptive message than would be possible with *errcode* alone. An implementation might, for
58579 example, save the character offset of the offending character of the pattern in a field of *preg*, and
58580 then include that in the generated message string. The implementation may also ignore *preg*.

58581 A REG_FILENAME flag was considered, but omitted. This flag caused *regexexec()* to match
 58582 patterns as described in XCU [Section 2.13](#) (on page 2382) instead of REs. This service is now
 58583 provided by the *fnmatch()* function.

58584 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and
 58585 POSIX.1-2017 in how to handle a “bad” regular expression. The ISO POSIX-2:1993 standard says
 58586 that many bad constructs “produce undefined results”, or that “the interpretation is undefined”.
 58587 POSIX.1-2017, however, says that the interpretation of such REs is unspecified. The term
 58588 “undefined” means that the action by the application is an error, of similar severity to passing a
 58589 bad pointer to a function.

58590 The *regcomp()* and *regexexec()* functions are required to accept any null-terminated string as the
 58591 *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is
 58592 “unspecified”. POSIX.1-2017 does not specify how the functions will interpret the pattern; they
 58593 might return error codes, or they might do pattern matching in some completely unexpected
 58594 way, but they should not do something like abort the process.

58595 FUTURE DIRECTIONS

58596 None.

58597 SEE ALSO

58598 [*fnmatch\(\)*](#), [*glob\(\)*](#)

58599 XBD [Chapter 9](#) (on page 181), [`<regex.h>`](#), [`<sys/types.h>`](#)

58600 XCU [Section 2.13](#) (on page 2382)

58601 CHANGE HISTORY

58602 First released in Issue 4. Derived from the ISO POSIX-2 standard.

58603 Issue 5

58604 Moved from POSIX2 C-language Binding to BASE.

58605 Issue 6

58606 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

58607 The following new requirements on POSIX implementations derive from alignment with the
 58608 Single UNIX Specification:

58609 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
 58610 required for conforming implementations of previous POSIX specifications, it was not
 58611 required for UNIX applications.

58612 The normative text is updated to avoid use of the term “must” for application requirements.

58613 The REG_ENOSYS constant is removed.

58614 The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexexec()* prototypes for
 58615 alignment with the ISO/IEC 9899:1999 standard.

58616 Issue 7

58617 Austin Group Interpretation 1003.1-2001 #134 is applied, clarifying that if more than one error
 58618 occurs in processing a function call, any one of the possible constants may be returned.

58619 SD5-XBD-ERN-60 is applied.

58620 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0505 [305] is applied.

58621 **NAME**

58622 remainder, remainderf, remainderl — remainder function

58623 **SYNOPSIS**

58624 #include <math.h>

58625 double remainder(double x, double y);

58626 float remainderf(float x, float y);

58627 long double remainderl(long double x, long double y);

58628 **DESCRIPTION**

58629 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 58630 conflict between the requirements described here and the ISO C standard is unintentional. This
 58631 volume of POSIX.1-2017 defers to the ISO C standard.

58632 These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value
 58633 n is the integral value nearest the exact value x/y . When $|n-x/y| = \frac{1}{2}$, the value is chosen to
 58634 be even.

58635 The behavior of *remainder()* shall be independent of the rounding mode.58636 **RETURN VALUE**58637 Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$
58638 when y is non-zero.58639 On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is
58640 implementation-defined whether a domain error occurs or zero is returned.58641 MX If x or y is NaN, a NaN shall be returned.58642 If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and a NaN shall be
58643 returned.58644 **ERRORS**

58645 These functions shall fail if:

58646 MX **Domain Error** The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-
58647 NaN.58648 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
58649 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
58650 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
58651 shall be raised.

58652 These functions may fail if:

58653 **Domain Error** The y argument is zero.58654 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
58655 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
58656 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
58657 shall be raised.

58658 **EXAMPLES**

58659 None.

58660 **APPLICATION USAGE**

58661 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
58662 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

58663 **RATIONALE**

58664 None.

58665 **FUTURE DIRECTIONS**

58666 None.

58667 **SEE ALSO**58668 [abs\(\)](#), [div\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ldiv\(\)](#)58669 XBD [Section 4.20](#) (on page 117), [<math.h>](#)58670 **CHANGE HISTORY**

58671 First released in Issue 4, Version 2.

58672 **Issue 5**

58673 Moved from X/OPEN UNIX extension to BASE.

58674 **Issue 6**58675 The *remainder()* function is no longer marked as an extension.

58676 The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999
58677 standard.

58678 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
58679 revised to align with the ISO/IEC 9899:1999 standard.

58680 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
58681 marked.

58682 **Issue 7**

58683 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

58684 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0506 [320] is applied.

58685 **NAME**

58686 remove — remove a file

58687 **SYNOPSIS**

```
58688 #include <stdio.h>
58689 int remove(const char *path);
```

58690 **DESCRIPTION**

58691 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 58692 conflict between the requirements described here and the ISO C standard is unintentional. This
 58693 volume of POSIX.1-2017 defers to the ISO C standard.

58694 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no
 58695 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,
 58696 unless it is created anew.

58697 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

58698 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

58699 **RETURN VALUE**

58700 CX Refer to *rmdir()* or *unlink()*.

58701 **ERRORS**

58702 CX Refer to *rmdir()* or *unlink()*.

58703 **EXAMPLES**58704 **Removing Access to a File**

58705 The following example shows how to remove access to a file named `/home/cnd/old_mods`.

```
58706 #include <stdio.h>
58707 int status;
58708 ...
58709 status = remove("/home/cnd/old_mods");
```

58710 **APPLICATION USAGE**

58711 None.

58712 **RATIONALE**

58713 None.

58714 **FUTURE DIRECTIONS**

58715 None.

58716 **SEE ALSO**

58717 *rmdir()*, *unlink()*

58718 XBD `<stdio.h>`

58719 **CHANGE HISTORY**

58720 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C
 58721 standard.

58722 **Issue 6**

58723 Extensions beyond the ISO C standard are marked.

58724
58725

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

58726
58727
58728

The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to *rmdir()*.

58729 **NAME**

58730 remque — remove an element from a queue

58731 **SYNOPSIS**

```
58732 XSI #include <search.h>  
58733 void remque(void *element);
```

58734 **DESCRIPTION**58735 Refer to *insque()*.

58736 **NAME**

58737 remquo, remquof, remquol — remainder functions

58738 **SYNOPSIS**

58739 #include <math.h>

58740 double remquo(double x, double y, int *quo);

58741 float remquof(float x, float y, int *quo);

58742 long double remquol(long double x, long double y, int *quo);

58743 **DESCRIPTION**

58744 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 58745 conflict between the requirements described here and the ISO C standard is unintentional. This
 58746 volume of POSIX.1-2017 defers to the ISO C standard.

58747 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the
 58748 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by *quo*,
 58749 they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo 2^n
 58750 to the magnitude of the integral quotient of x/y , where n is an implementation-defined integer
 58751 greater than or equal to 3. If y is zero, the value stored in the object pointed to by *quo* is
 58752 unspecified.

58753 An application wishing to check for error situations should set *errno* to zero and call
 58754 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 58755 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 58756 zero, an error has occurred.

58757 **RETURN VALUE**58758 These functions shall return $x \text{ REM } y$.

58759 On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is
 58760 implementation-defined whether a domain error occurs or zero is returned.

58761 MX If x or y is NaN, a NaN shall be returned.

58762 If x is $\pm\text{Inf}$ or y is zero and the other argument is non-NaN, a domain error shall occur, and a
 58763 NaN shall be returned.

58764 **ERRORS**

58765 These functions shall fail if:

58766 MX **Domain Error** The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-
 58767 NaN.

58768 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 58769 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 58770 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 58771 shall be raised.

58772 These functions may fail if:

58773 **Domain Error** The y argument is zero.

58774 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 58775 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 58776 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 58777 shall be raised.

58778 **EXAMPLES**

58779 None.

58780 **APPLICATION USAGE**

58781 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
58782 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

58783 **RATIONALE**

58784 These functions are intended for implementing argument reductions which can exploit a few
58785 low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an
58786 exact representation of the quotient is not practical.

58787 **FUTURE DIRECTIONS**

58788 None.

58789 **SEE ALSO**58790 [fclearexcept\(\)](#), [fetestexcept\(\)](#), [remainder\(\)](#)58791 XBD [Section 4.20](#) (on page 117), [<math.h>](#)58792 **CHANGE HISTORY**

58793 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

58794 **Issue 7**

58795 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

58796 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0507 [320] is applied.

58797 **NAME**

58798 rename, renameat — rename file

58799 **SYNOPSIS**

58800 #include <stdio.h>

58801 int rename(const char *old, const char *new);

58802 OH CX #include <fcntl.h>

58803 CX int renameat(int oldfd, const char *old, int newfd,
58804 const char *new);58805 **DESCRIPTION**58806 CX For *rename()*: The functionality described on this reference page is aligned with the ISO C
58807 standard. Any conflict between the requirements described here and the ISO C standard is
58808 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.58809 The *rename()* function shall change the name of a file. The *old* argument points to the pathname
58810 CX of the file to be renamed. The *new* argument points to the new pathname of the file. If the *new*
58811 argument does not resolve to an existing directory entry for a file of type directory and the *new*
58812 argument contains at least one non-`<slash>` character and ends with one or more trailing
58813 `<slash>` characters after all symbolic links have been processed, *rename()* shall fail.58814 If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic
58815 link itself, and shall not resolve the last component of the argument. If the *old* argument and the
58816 *new* argument resolve to either the same existing directory entry or different directory entries for
58817 the same existing file, *rename()* shall return successfully and perform no other action.58818 If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall
58819 not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be
58820 removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other
58821 threads throughout the renaming operation and refer either to the file referred to by *new* or *old*
58822 before the operation began. Write access permission is required for both the directory containing
58823 *old* and the directory containing *new*.58824 If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the
58825 pathname of a file that is not a directory. If the directory named by the *new* argument exists, it
58826 shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout
58827 the renaming operation and shall refer either to the directory referred to by *new* or *old* before the
58828 operation began. If *new* names an existing directory, it shall be required to be an empty directory.58829 If either *pathname* argument refers to a path whose final component is either dot or dot-dot,
58830 *rename()* shall fail.58831 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.
58832 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.58833 The *old* pathname shall not name an ancestor directory of the *new* pathname. Write access
58834 permission is required for the directory containing *old* and the directory containing *new*. If the
58835 *old* argument points to the pathname of a directory, write access permission may be required for
58836 the directory named by *old*, and, if it exists, the directory named by *new*.58837 If the link named by the *new* argument exists and the file's link count becomes 0 when it is
58838 removed and no process has the file open, the space occupied by the file shall be freed and the
58839 file shall no longer be accessible. If one or more processes have the file open when the last link is
58840 removed, the link shall be removed before *rename()* returns, but the removal of the file contents
58841 shall be postponed until all references to the file are closed.

58842 Upon successful completion, *rename()* shall mark for update the last data modification and last
58843 file status change timestamps of the parent directory of each file.

58844 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be
58845 unaffected.

58846 The *renameat()* function shall be equivalent to the *rename()* function except in the case where
58847 either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located
58848 relative to the directory associated with the file descriptor *oldfd* instead of the current working
58849 directory. If *new* is a relative path, the same happens only relative to the directory associated
58850 with *newfd*. If the access mode of the open file description associated with the file descriptor is
58851 not O_SEARCH, the function shall check whether directory searches are permitted using the
58852 current permissions of the directory underlying the file descriptor. If the access mode is
58853 O_SEARCH, the function shall not perform the check.

58854 If *renameat()* is passed the special value AT_FDCWD in the *oldfd* or *newfd* parameter, the current
58855 working directory shall be used in the determination of the file for the respective *path* parameter.

58856 RETURN VALUE

58857 CX Upon successful completion, the *rename()* function shall return 0. Otherwise, it shall return `-1`,
58858 *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by
58859 *new* shall be changed or created.

58860 CX Upon successful completion, the *renameat()* function shall return 0. Otherwise, it shall return `-1`
58861 and set *errno* to indicate the error.

58862 ERRORS

58863 CX The *rename()* and *renameat()* functions shall fail if:

58864 CX [EACCES] A component of either path prefix denies search permission; or one of the
58865 directories containing *old* or *new* denies write permissions; or, write
58866 permission is required and is denied for a directory pointed to by the *old* or
58867 *new* arguments.

58868 CX [EBUSY] The directory named by *old* or *new* is currently in use by the system or another
58869 process, and the implementation considers this an error.

58870 CX [EEXIST] or [ENOTEMPTY]
58871 The link named by *new* is a directory that is not an empty directory.

58872 CX [EINVAL] The *old* pathname names an ancestor directory of the *new* pathname, or either
58873 pathname argument contains a final component that is dot or dot-dot.

58874 CX [EIO] A physical I/O error has occurred.

58875 CX [EISDIR] The *new* argument points to a directory and the *old* argument points to a file
58876 that is not a directory.

58877 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
58878 argument.

58879 CX [EMLINK] The file named by *old* is a directory, and the link count of the parent directory
58880 of *new* would exceed {LINK_MAX}.

58881 CX [ENAMETOOLONG]
58882 The length of a component of a pathname is longer than {NAME_MAX}.

58883	CX	[ENOENT]	The link named by <i>old</i> does not name an existing file, a component of the path prefix of <i>new</i> does not exist, or either <i>old</i> or <i>new</i> points to an empty string.
58884			
58885	CX	[ENOSPC]	The directory that would contain <i>new</i> cannot be extended.
58886	CX	[ENOTDIR]	A component of either path prefix names an existing file that is neither a directory nor a symbolic link to a directory; or the <i>old</i> argument names a directory and the <i>new</i> argument names a non-directory file; or the <i>old</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or the <i>old</i> argument names an existing non-directory file and the <i>new</i> argument names a nonexistent file, contains at least one non- <code><slash></code> character, and ends with one or more trailing <code><slash></code> characters; or the <i>new</i> argument names an existing non-directory file, contains at least one non- <code><slash></code> character, and ends with one or more trailing <code><slash></code> characters.
58887			
58888			
58889			
58890			
58891			
58892			
58893			
58894			
58895			
58896			
58897	XSI	[EPERM] or [EACCES]	The <code>S_ISVTX</code> flag is set on the directory containing the file referred to by <i>old</i> and the process does not satisfy the criteria specified in XBD Section 4.3 (on page 108) with respect to <i>old</i> ; or <i>new</i> refers to an existing file, the <code>S_ISVTX</code> flag is set on the directory containing this file, and the process does not satisfy the criteria specified in XBD Section 4.3 with respect to this file.
58898			
58899			
58900			
58901			
58902			
58903	CX	[EROFS]	The requested operation requires writing in a directory on a read-only file system.
58904			
58905	CX	[EXDEV]	The links named by <i>new</i> and <i>old</i> are on different file systems and the implementation does not support links between file systems.
58906			
58907	CX		In addition, the <code>renameat()</code> function shall fail if:
58908		[EACCES]	The access mode of the open file description associated with <i>olddfd</i> or <i>newfd</i> is not <code>O_SEARCH</code> and the permissions of the directory underlying <i>olddfd</i> or <i>newfd</i> , respectively, do not permit directory searches.
58909			
58910			
58911		[EBADF]	The <i>old</i> argument does not specify an absolute path and the <i>olddfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching, or the <i>new</i> argument does not specify an absolute path and the <i>newfd</i> argument is neither <code>AT_FDCWD</code> nor a valid file descriptor open for reading or searching.
58912			
58913			
58914			
58915			
58916		[ENOTDIR]	The <i>old</i> or <i>new</i> argument is not an absolute path and <i>olddfd</i> or <i>newfd</i> , respectively, is a file descriptor associated with a non-directory file.
58917			
58918	CX		The <code>rename()</code> and <code>renameat()</code> functions may fail if:
58919	OB XSR	[EBUSY]	The file named by the <i>old</i> or <i>new</i> arguments is a named STREAM.
58920	CX	[ELOOP]	More than <code>{SYMLOOP_MAX}</code> symbolic links were encountered during resolution of the <i>path</i> argument.
58921			
58922	CX	[ENAMETOOLONG]	The length of a pathname exceeds <code>{PATH_MAX}</code> , or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds <code>{PATH_MAX}</code> .
58923			
58924			
58925			

58926 CX [ETXTBSY] The file named by *new* exists and is the last directory entry to a pure procedure
 58927 (shared text) file that is being executed.

58928 EXAMPLES

58929 Renaming a File

58930 The following example shows how to rename a file named `/home/cnd/mod1` to
 58931 `/home/cnd/mod2`.

```
58932 #include <stdio.h>
58933 int status;
58934 ...
58935 status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

58936 APPLICATION USAGE

58937 Some implementations mark for update the last file status change timestamp of renamed files
 58938 and some do not. Applications which make use of the last file status change timestamp may
 58939 behave differently with respect to renamed files unless they are designed to allow for either
 58940 behavior.

58941 RATIONALE

58942 This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its
 58943 inclusion here expands that definition to include actions on directories and specifies behavior
 58944 when the *new* parameter names a file that already exists. That specification requires that the
 58945 action of the function be atomic.

58946 One of the reasons for introducing this function was to have a means of renaming directories
 58947 while permitting implementations to prohibit the use of *link()* and *unlink()* with directories,
 58948 thus constraining links to directories to those made by *mkdir()*.

58949 The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
58950 rename("x", "x");
```

58951 does not remove the file.

58952 Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

58953 See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir()* and [EBUSY] in
 58954 *unlink()*. For a discussion of [EXDEV], see *link()*.

58955 The purpose of the *renameat()* function is to rename files in directories other than the current
 58956 working directory without exposure to race conditions. Any part of the path of a file could be
 58957 changed in parallel to a call to *rename()*, resulting in unspecified behavior. By opening file
 58958 descriptors for the source and target directories and using the *renameat()* function it can be
 58959 guaranteed that that renamed file is located correctly and the resulting file is in the desired
 58960 directory.

58961 FUTURE DIRECTIONS

58962 None.

58963 SEE ALSO

58964 [link\(\)](#), [rmdir\(\)](#), [symlink\(\)](#), [unlink\(\)](#)

58965 XBD Section 4.3 (on page 108), [<fcntl.h>](#), [<stdio.h>](#)

58966 **CHANGE HISTORY**

58967 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

58968 **Issue 5**

58969 The [EBUSY] error is added to the optional part of the ERRORS section.

58970 **Issue 6**

58971 Extensions beyond the ISO C standard are marked.

58972 The following new requirements on POSIX implementations derive from alignment with the
58973 Single UNIX Specification:

58974 The [EIO] mandatory error condition is added.

58975 The [ELOOP] mandatory error condition is added.

58976 A second [ENAMETOOLONG] is added as an optional error condition.

58977 The [ETXTBSY] optional error condition is added.

58978 The following changes were made to align with the IEEE P1003.1a draft standard:

58979 Details are added regarding the treatment of symbolic links.

58980 The [ELOOP] optional error condition is added.

58981 The normative text is updated to avoid use of the term “must” for application requirements.

58982 **Issue 7**

58983 Austin Group Interpretation 1003.1-2001 #016 is applied, changing the definition of the
58984 [ENOTDIR] error.

58985 Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final
58986 component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.

58987 Austin Group Interpretation 1003.1-2001 #143 is applied.

58988 Austin Group Interpretation 1003.1-2001 #145 is applied, clarifying that the [ENOENT] error
58989 condition also applies to the case in which a component of *new* does not exist.

58990 Austin Group Interpretations 1003.1-2001 #174 and #181 are applied.

58991 The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API
58992 Set Part 2.

58993 Changes are made related to support for finegrained timestamps.

58994 Changes are made to allow a directory to be opened for searching.

58995 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0508 [324], XSH/TC1-2008/0509 [147],
58996 XSH/TC1-2008/0510 [379], XSH/TC1-2008/0511 [278], and XSH/TC1-2008/0512 [278] are
58997 applied.

58998 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0311 [873], XSH/TC2-2008/0312 [591],
58999 XSH/TC2-2008/0313 [716], XSH/TC2-2008/0314 [817], XSH/TC2-2008/0315 [817], and
59000 XSH/TC2-2008/0316 [591] are applied.

59001 **NAME**59002 `rewind` — reset the file position indicator in a stream59003 **SYNOPSIS**59004 `#include <stdio.h>`59005 `void rewind(FILE *stream);`59006 **DESCRIPTION**59007 CX The functionality described on this reference page is aligned with the ISO C standard. Any
59008 conflict between the requirements described here and the ISO C standard is unintentional. This
59009 volume of POSIX.1-2017 defers to the ISO C standard.

59010 The call:

59011 `rewind(stream)`

59012 shall be equivalent to:

59013 `(void) fseek(stream, 0L, SEEK_SET)`59014 except that `rewind()` shall also clear the error indicator.59015 CX Since `rewind()` does not return a value, an application wishing to detect errors should clear `errno`,
59016 then call `rewind()`, and if `errno` is non-zero, assume an error has occurred.59017 **RETURN VALUE**59018 The `rewind()` function shall not return a value.59019 **ERRORS**59020 CX Refer to `fseek()` with the exception of [EINVAL] which does not apply.59021 **EXAMPLES**

59022 None.

59023 **APPLICATION USAGE**

59024 None.

59025 **RATIONALE**

59026 None.

59027 **FUTURE DIRECTIONS**

59028 None.

59029 **SEE ALSO**59030 [Section 2.5](#) (on page 495), `fseek()`59031 XBD `<stdio.h>`59032 **CHANGE HISTORY**

59033 First released in Issue 1. Derived from Issue 1 of the SVID.

59034 **Issue 6**

59035 Extensions beyond the ISO C standard are marked.

59036 **Issue 7**

59037 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0513 [14] is applied.

59038 **NAME**59039 `rewinddir` — reset the position of a directory stream to the beginning of a directory59040 **SYNOPSIS**59041 `#include <dirent.h>`59042 `void rewinddir(DIR *dirp);`59043 **DESCRIPTION**

59044 The `rewinddir()` function shall reset the position of the directory stream to which `dirp` refers to the
59045 beginning of the directory. It shall also cause the directory stream to refer to the current state of
59046 the corresponding directory, as a call to `opendir()` would have done. If `dirp` does not refer to a
59047 directory stream, the effect is undefined.

59048 After a call to the `fork()` function, either the parent or child (but not both) may continue
59049 XSI processing the directory stream using `readdir()`, `rewinddir()`, or `seekdir()`. If both the parent and
59050 child processes use these functions, the result is undefined.

59051 **RETURN VALUE**59052 The `rewinddir()` function shall not return a value.59053 **ERRORS**

59054 No errors are defined.

59055 **EXAMPLES**

59056 None.

59057 **APPLICATION USAGE**

59058 The `rewinddir()` function should be used in conjunction with `opendir()`, `readdir()`, and `closedir()` to
59059 examine the contents of the directory. This method is recommended for portability.

59060 **RATIONALE**

59061 None.

59062 **FUTURE DIRECTIONS**

59063 None.

59064 **SEE ALSO**59065 [*closedir\(\)*](#), [*fdopendir\(\)*](#), [*readdir\(\)*](#)59066 XBD [*<dirent.h>*](#), [*<sys/types.h>*](#)59067 **CHANGE HISTORY**

59068 First released in Issue 2.

59069 **Issue 6**59070 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

59071 The following new requirements on POSIX implementations derive from alignment with the
59072 Single UNIX Specification:

59073 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
59074 required for conforming implementations of previous POSIX specifications, it was not
59075 required for UNIX applications.

59076 **NAME**

59077 rint, rintf, rintl — round-to-nearest integral value

59078 **SYNOPSIS**

```
59079 #include <math.h>
59080 double rint(double x);
59081 float rintf(float x);
59082 long double rintl(long double x);
```

59083 **DESCRIPTION**

59084 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 59085 conflict between the requirements described here and the ISO C standard is unintentional. This
 59086 volume of POSIX.1-2017 defers to the ISO C standard.

59087 These functions shall return the integral value (represented as a **double**) nearest x in the
 59088 direction of the current rounding mode. The current rounding mode is implementation-defined.

59089 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to
 59090 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be
 59091 equivalent to *ceil()*. If the current rounding mode rounds towards zero, then *rint()* shall be
 59092 MX equivalent to *trunc()*. If the current rounding mode rounds towards nearest, then *rint()* differs
 59093 from *round()* in that halfway cases are rounded to even rather than away from zero.

59094 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that
 59095 they may raise the inexact floating-point exception if the result differs in value from the
 59096 argument.

59097 An application wishing to check for error situations should set *errno* to zero and call
 59098 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 59099 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 59100 zero, an error has occurred.

59101 **RETURN VALUE**

59102 Upon successful completion, these functions shall return the integer (represented as a double
 59103 MX precision number) nearest x in the direction of the current rounding mode. The result shall have
 59104 the same sign as x .

59105 MX If x is NaN, a NaN shall be returned.

59106 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

59107 **ERRORS**

59108 No errors are defined.

59109 **EXAMPLES**

59110 None.

59111 **APPLICATION USAGE**

59112 The integral value returned by these functions need not be expressible as an **intmax_t**. The
 59113 return value should be tested before assigning it to an integer type to avoid the undefined
 59114 results of an integer overflow.

59115 **RATIONALE**

59116 None.

59117 **FUTURE DIRECTIONS**

59118 None.

59119 **SEE ALSO**59120 *abs()*, *ceil()*, *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, *nearbyint()*59121 XBD Section 4.20 (on page 117), **<math.h>**59122 **CHANGE HISTORY**

59123 First released in Issue 4, Version 2.

59124 **Issue 5**

59125 Moved from X/OPEN UNIX extension to BASE.

59126 **Issue 6**

59127 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

59128 The *rintf()* and *rintl()* functions are added.59129 The *rint()* function is no longer marked as an extension.59130 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
59131 revised to align with the ISO/IEC 9899:1999 standard.59132 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard
59133 are marked.59134 **Issue 7**59135 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0514 [346], XSH/TC1-2008/0515 [346],
59136 XSH/TC1-2008/0516 [346], XSH/TC1-2008/0517 [346], and XSH/TC1-2008/0518 [346] are
59137 applied.

59138 **NAME**

59139 rmdir — remove a directory

59140 **SYNOPSIS**

59141 #include <unistd.h>

59142 int rmdir(const char *path);

59143 **DESCRIPTION**59144 The *rmdir()* function shall remove a directory whose name is given by *path*. The directory shall
59145 be removed only if it is an empty directory.59146 If the directory is the root directory or the current working directory of any process, it is
59147 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].59148 If *path* names a symbolic link, then *rmdir()* shall fail and set *errno* to [ENOTDIR].59149 If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir()* shall
59150 fail.59151 If the directory's link count becomes 0 and no process has the directory open, the space occupied
59152 by the directory shall be freed and the directory shall no longer be accessible. If one or more
59153 processes have the directory open when the last link is removed, the dot and dot-dot entries, if
59154 present, shall be removed before *rmdir()* returns and no new entries may be created in the
59155 directory, but the directory shall not be removed until all references to the directory are closed.59156 If the directory is not an empty directory, *rmdir()* shall fail and set *errno* to [EEXIST] or
59157 [ENOTEMPTY].59158 Upon successful completion, *rmdir()* shall mark for update the last data modification and last
59159 file status change timestamps of the parent directory.59160 **RETURN VALUE**59161 Upon successful completion, the function *rmdir()* shall return 0. Otherwise, -1 shall be returned,
59162 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.59163 **ERRORS**59164 The *rmdir()* function shall fail if:59165 [EACCES] Search permission is denied on a component of the path prefix, or write
59166 permission is denied on the parent directory of the directory to be removed.59167 [EBUSY] The directory to be removed is currently in use by the system or some process
59168 and the implementation considers this to be an error.59169 [EEXIST] or [ENOTEMPTY] The *path* argument names a directory that is not an empty directory, or there
59170 are hard links to the directory other than dot or a single entry in dot-dot.
5917159172 [EINVAL] The *path* argument contains a last component that is dot.

59173 [EIO] A physical I/O error has occurred.

59174 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
59175 argument.59176 [ENAMETOOLONG] The length of a component of a pathname is longer than {NAME_MAX}.
5917759178 [ENOENT] A component of *path* does not name an existing file, or the *path* argument
59179 names a nonexistent directory or points to an empty string.

- 59180 [ENOTDIR] A component of *path* names an existing file that is neither a directory nor a
59181 symbolic link to a directory.
- 59182 XSI [EPERM] or [EACCES]
59183 The S_ISVTX flag is set on the directory containing the file referred to by the
59184 *path* argument and the process does not satisfy the criteria specified in XBD
59185 [Section 4.3](#) (on page 108).
- 59186 [EROFS] The directory entry to be removed resides on a read-only file system.
- 59187 The *rmdir()* function may fail if:
- 59188 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
59189 resolution of the *path* argument.
- 59190 [ENAMETOOLONG]
59191 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
59192 symbolic link produced an intermediate result with a length that exceeds
59193 {PATH_MAX}.

59194 EXAMPLES

59195 Removing a Directory

59196 The following example shows how to remove a directory named `/home/cnd/mod1`.

```
59197 #include <unistd.h>
59198 int status;
59199 ...
59200 status = rmdir("/home/cnd/mod1");
```

59201 APPLICATION USAGE

59202 None.

59203 RATIONALE

59204 The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the
59205 condition when the directory to be removed does not exist or *new* already exists. When the 1984
59206 /usr/group standard was published, it contained [EEXIST] instead. When these functions were
59207 adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore,
59208 several existing applications and implementations support/use both forms, and no agreement
59209 could be reached on either value. All implementations are required to supply both [EEXIST] and
59210 [ENOTEMPTY] in `<errno.h>` with distinct values, so that applications can use both values in C-
59211 language **case** statements.

59212 The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the
59213 parent directory to be removed is not clear, particularly in the presence of multiple links to a
59214 directory.

59215 The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are
59216 multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or
59217 [ENOTEMPTY] clarifies the behavior in this case.

59218 If the current working directory of the process is being removed, that should be an allowed
59219 error.

59220 Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in
59221 [Section 2.3](#) (on page 481) about returning any one of the possible errors permits that behavior to
59222 continue. The [ELOOP] error may be returned if more than {SYMLOOP_MAX} symbolic links

59223 are encountered during resolution of the *path* argument.

59224 **FUTURE DIRECTIONS**

59225 None.

59226 **SEE ALSO**

59227 [Section 2.3](#) (on page 481), *mkdir()*, *remove()*, *rename()*, *unlink()*

59228 [XBD Section 4.3](#) (on page 108), [<unistd.h>](#)

59229 **CHANGE HISTORY**

59230 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

59231 **Issue 6**

59232 The following new requirements on POSIX implementations derive from alignment with the
59233 Single UNIX Specification:

59234 The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.

59235 The [EIO] mandatory error condition is added.

59236 The [ELOOP] mandatory error condition is added.

59237 A second [ENAMETOOLONG] is added as an optional error condition.

59238 The following changes were made to align with the IEEE P1003.1a draft standard:

59239 The [ELOOP] optional error condition is added.

59240 **Issue 7**

59241 Austin Group Interpretation 1003.1-2001 #143 is applied.

59242 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for
59243 operations when the S_ISVTX bit is set.

59244 Changes are made related to support for finegrained timestamps.

59245 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0519 [324] is applied.

59246 **NAME**

59247 round, roundf, roundl — round to the nearest integer value in a floating-point format

59248 **SYNOPSIS**

```
59249 #include <math.h>
59250 double round(double x);
59251 float roundf(float x);
59252 long double roundl(long double x);
```

59253 **DESCRIPTION**

59254 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 59255 conflict between the requirements described here and the ISO C standard is unintentional. This
 59256 volume of POSIX.1-2017 defers to the ISO C standard.

59257 These functions shall round their argument to the nearest integer value in floating-point format,
 59258 rounding halfway cases away from zero, regardless of the current rounding direction.

59259 **RETURN VALUE**

59260 MX Upon successful completion, these functions shall return the rounded integer value. The result
 59261 shall have the same sign as x .

59262 MX If x is NaN, a NaN shall be returned.

59263 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

59264 **ERRORS**

59265 No errors are defined.

59266 **EXAMPLES**

59267 None.

59268 **APPLICATION USAGE**

59269 The integral value returned by these functions need not be expressible as an `intmax_t`. The
 59270 return value should be tested before assigning it to an integer type to avoid the undefined
 59271 results of an integer overflow.

59272 These functions may raise the inexact floating-point exception if the result differs in value from
 59273 the argument.

59274 **RATIONALE**

59275 None.

59276 **FUTURE DIRECTIONS**

59277 None.

59278 **SEE ALSO**

59279 [fclearexcept\(\)](#), [fetestexcept\(\)](#)

59280 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

59281 **CHANGE HISTORY**

59282 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

59283 **Issue 7**

59284 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0520 [346] is applied.

59285 **NAME**

59286 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT_RADIX

59287 **SYNOPSIS**

```
59288 #include <math.h>
59289 double scalbn(double x, long n);
59290 float scalblnf(float x, long n);
59291 long double scalblnl(long double x, long n);
59292 double scalbn(double x, int n);
59293 float scalbnf(float x, int n);
59294 long double scalbnl(long double x, int n);
```

59295 **DESCRIPTION**

59296 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 59297 conflict between the requirements described here and the ISO C standard is unintentional. This
 59298 volume of POSIX.1-2017 defers to the ISO C standard.

59299 These functions shall compute $x * FLT_RADIX^n$ efficiently, not normally by computing
 59300 FLT_RADIX^n explicitly.

59301 An application wishing to check for error situations should set *errno* to zero and call
 59302 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 59303 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 59304 zero, an error has occurred.

59305 **RETURN VALUE**

59306 Upon successful completion, these functions shall return $x * FLT_RADIX^n$.

59307 If the result would cause overflow, a range error shall occur and these functions shall return
 59308 $\pm HUGE_VAL$, $\pm HUGE_VALF$, and $\pm HUGE_VALL$ (according to the sign of *x*) as appropriate for
 59309 the return type of the function.

59310 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 59311 MXX and *scalbn()*, *scalblnf()*, *scalblnl()*, *scalbn()*, *scalbnf()*, and *scalbnl()* shall return 0.0, or (if IEC
 59312 60559 Floating-Point is not supported) an implementation-defined value no greater in
 59313 magnitude than DBL_MIN, FLT_MIN, LDBL_MIN, DBL_MIN, FLT_MIN, and LDBL_MIN,
 59314 respectively.

59315 MX If *x* is NaN, a NaN shall be returned.

59316 If *x* is ± 0 or $\pm Inf$, *x* shall be returned.

59317 If *n* is 0, *x* shall be returned.

59318 MXX If the correct value would cause underflow, and is representable, a range error may occur and
 59319 the correct value shall be returned.

59320 **ERRORS**

59321 These functions shall fail if:

59322 Range Error The result overflows.

59323 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 59324 then *errno* shall be set to [ERANGE]. If the integer expression
 59325 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 59326 floating-point exception shall be raised.

59327 These functions may fail if:

59328 Range Error The result underflows.

59329 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
59330 then *errno* shall be set to [ERANGE]. If the integer expression
59331 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
59332 floating-point exception shall be raised.

59333 EXAMPLES

59334 None.

59335 APPLICATION USAGE

59336 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
59337 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

59338 RATIONALE

59339 These functions are named so as to avoid conflicting with the historical definition of the *scalb()*
59340 function from the Single UNIX Specification. The difference is that the *scalb()* function has a
59341 second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard.
59342 The three functions whose second type is **long** are provided because the factor required to scale
59343 from the smallest positive floating-point value to the largest finite one, on many
59344 implementations, is too large to represent in the minimum-width **int** format.

59345 FUTURE DIRECTIONS

59346 None.

59347 SEE ALSO

59348 *feclearexcept()*, *fetestexcept()*

59349 XBD Section 4.20 (on page 117), [<math.h>](#)

59350 CHANGE HISTORY

59351 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

59352 Issue 7

59353 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0521 [68] and XSH/TC1-2008/0522
59354 [68] are applied.

59355 **NAME**

59356 scandir — scan a directory

59357 **SYNOPSIS**

59358 #include <dirent.h>

```
59359       int scandir(const char *dir, struct dirent ***namelist,  
59360                   int (*sel)(const struct dirent *),  
59361                   int (*compar)(const struct dirent **, const struct dirent **));
```

59362 **DESCRIPTION**59363 Refer to *alphasort()*.

59364 **NAME**

59365 scanf ‡convert formatted input

59366 **SYNOPSIS**

59367 #include <stdio.h>

59368 int scanf(const char *restrict *format*, ...);

59369 **DESCRIPTION**

59370 Refer to [fscanf\(\)](#).

59371 **NAME**59372 sched_get_priority_max, sched_get_priority_min ‡get priority limits **REALTIME**)59373 **SYNOPSIS**

```
59374 PS|TPS #include <sched.h>
59375 int sched_get_priority_max(int policy);
59376 int sched_get_priority_min(int policy);
```

59377 **DESCRIPTION**

59378 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the appropriate
 59379 maximum or minimum, respectively, for the scheduling policy specified by *policy*.

59380 The value of *policy* shall be one of the scheduling policy values defined in **<sched.h>**.

59381 **RETURN VALUE**

59382 If successful, the *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the
 59383 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a
 59384 value of -1 and set *errno* to indicate the error.

59385 **ERRORS**

59386 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall fail if:

59387 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling
 59388 policy.

59389 **EXAMPLES**

59390 None.

59391 **APPLICATION USAGE**

59392 None.

59393 **RATIONALE**

59394 None.

59395 **FUTURE DIRECTIONS**

59396 None.

59397 **SEE ALSO**

59398 [sched_getparam\(\)](#), [sched_setparam\(\)](#), [sched_getscheduler\(\)](#), [sched_rr_get_interval\(\)](#),
 59399 [sched_setscheduler\(\)](#)

59400 XBD **<sched.h>**

59401 **CHANGE HISTORY**

59402 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59403 **Issue 6**

59404 These functions are marked as part of the Process Scheduling option.

59405 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59406 implementation does not support the Process Scheduling option.

59407 The [ESRCH] error condition has been removed since these functions do not take a *pid*
 59408 argument.

59409 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin
 59410 code in the SYNOPSIS to PS|TPS.

59411 **NAME**59412 sched_getparam ‡get scheduling parameters **REALTIME**)59413 **SYNOPSIS**

```
59414 PS #include <sched.h>
59415 int sched_getparam(pid_t pid, struct sched_param *param);
```

59416 **DESCRIPTION**

59417 The *sched_getparam()* function shall return the scheduling parameters of a process specified by
 59418 *pid* in the **sched_param** structure pointed to by *param*.

59419 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 59420 parameters for the process whose process ID is equal to *pid* shall be returned.

59421 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of
 59422 the *sched_getparam()* function is unspecified if the value of *pid* is negative.

59423 **RETURN VALUE**

59424 Upon successful completion, the *sched_getparam()* function shall return zero. If the call to
 59425 *sched_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate
 59426 the error.

59427 **ERRORS**

59428 The *sched_getparam()* function shall fail if:

59429 [EPERM] The requesting process does not have permission to obtain the scheduling
 59430 parameters of the specified process.

59431 [ESRCH] No process can be found corresponding to that specified by *pid*.

59432 **EXAMPLES**

59433 None.

59434 **APPLICATION USAGE**

59435 None.

59436 **RATIONALE**

59437 None.

59438 **FUTURE DIRECTIONS**

59439 None.

59440 **SEE ALSO**

59441 [sched_getscheduler\(\)](#), [sched_setparam\(\)](#), [sched_setscheduler\(\)](#)

59442 XBD [<sched.h>](#)

59443 **CHANGE HISTORY**

59444 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59445 **Issue 6**

59446 The *sched_getparam()* function is marked as part of the Process Scheduling option.

59447 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59448 implementation does not support the Process Scheduling option.

59449 **NAME**

59450 sched_getscheduler ‡get scheduling policy (REALTIME)

59451 **SYNOPSIS**

```
59452 PS #include <sched.h>
59453 int sched_getscheduler(pid_t pid);
```

59454 **DESCRIPTION**

59455 The *sched_getscheduler()* function shall return the scheduling policy of the process specified by
 59456 *pid*. If the value of *pid* is negative, the behavior of the *sched_getscheduler()* function is
 59457 unspecified.

59458 The values that can be returned by *sched_getscheduler()* are defined in the **<sched.h>** header.

59459 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 59460 policy shall be returned for the process whose process ID is equal to *pid*.

59461 If *pid* is zero, the scheduling policy shall be returned for the calling process.

59462 **RETURN VALUE**

59463 Upon successful completion, the *sched_getscheduler()* function shall return the scheduling policy
 59464 of the specified process. If unsuccessful, the function shall return -1 and set *errno* to indicate the
 59465 error.

59466 **ERRORS**

59467 The *sched_getscheduler()* function shall fail if:

59468 [EPERM] The requesting process does not have permission to determine the scheduling
 59469 policy of the specified process.

59470 [ESRCH] No process can be found corresponding to that specified by *pid*.

59471 **EXAMPLES**

59472 None.

59473 **APPLICATION USAGE**

59474 None.

59475 **RATIONALE**

59476 None.

59477 **FUTURE DIRECTIONS**

59478 None.

59479 **SEE ALSO**

59480 [sched_getparam\(\)](#), [sched_setparam\(\)](#), [sched_setscheduler\(\)](#)

59481 XBD **<sched.h>**

59482 **CHANGE HISTORY**

59483 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59484 **Issue 6**

59485 The *sched_getscheduler()* function is marked as part of the Process Scheduling option.

59486 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59487 implementation does not support the Process Scheduling option.

59488 **NAME**

59489 sched_rr_get_interval ‡get execution time limits(REALTIME)

59490 **SYNOPSIS**

```
59491 PS|TPS #include <sched.h>
59492 int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

59493 **DESCRIPTION**

59494 The *sched_rr_get_interval()* function shall update the **timespec** structure referenced by the
 59495 *interval* argument to contain the current execution time limit (that is, time quantum) for the
 59496 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process
 59497 shall be returned.

59498 **RETURN VALUE**

59499 If successful, the *sched_rr_get_interval()* function shall return zero. Otherwise, it shall return a
 59500 value of -1 and set *errno* to indicate the error.

59501 **ERRORS**

59502 The *sched_rr_get_interval()* function shall fail if:

59503 [ESRCH] No process can be found corresponding to that specified by *pid*.

59504 **EXAMPLES**

59505 None.

59506 **APPLICATION USAGE**

59507 None.

59508 **RATIONALE**

59509 None.

59510 **FUTURE DIRECTIONS**

59511 None.

59512 **SEE ALSO**

59513 [sched_getparam\(\)](#), [sched_get_priority_max\(\)](#), [sched_getscheduler\(\)](#), [sched_setparam\(\)](#),
 59514 [sched_setscheduler\(\)](#)

59515 XBD [<sched.h>](#)

59516 **CHANGE HISTORY**

59517 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59518 **Issue 6**

59519 The *sched_rr_get_interval()* function is marked as part of the Process Scheduling option.

59520 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59521 implementation does not support the Process Scheduling option.

59522 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in
 59523 the SYNOPSIS to PS|TPS.

59524 **NAME**

59525 sched_setparam ‡'set scheduling parameters(REALTIME)

59526 **SYNOPSIS**

```
59527 PS #include <sched.h>
59528 int sched_setparam(pid_t pid, const struct sched_param *param);
```

59529 **DESCRIPTION**

59530 The *sched_setparam()* function shall set the scheduling parameters of the process specified by *pid*
 59531 to the values specified by the **sched_param** structure pointed to by *param*. The value of the
 59532 *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive
 59533 priority range for the current scheduling policy of the process specified by *pid*. Higher
 59534 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the
 59535 behavior of the *sched_setparam()* function is unspecified.

59536 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 59537 parameters shall be set for the process whose process ID is equal to *pid*.

59538 If *pid* is zero, the scheduling parameters shall be set for the calling process.

59539 The conditions under which one process has permission to change the scheduling parameters of
 59540 another process are implementation-defined.

59541 Implementations may require the requesting process to have appropriate privileges to set its
 59542 own scheduling parameters or those of another process.

59543 See [Scheduling Policies](#) (on page 506) for a description on how this function affects the
 59544 scheduling of the threads within the target process.

59545 SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or
 59546 SCHED_SPORADIC, the result is implementation-defined; this case includes the
 59547 SCHED_OTHER policy.

59548 SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified
 59549 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

59550 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
 59551 function to succeed; if not, the function shall fail. It is unspecified whether the
 59552 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
 59553 rounded to align with the resolution of the clock being used.

59554 This function is not atomic with respect to other threads in the process. Threads may continue to
 59555 execute while this function call is in the process of changing the scheduling policy for the
 59556 underlying kernel-scheduled entities used by the process contention scope threads.

59557 **RETURN VALUE**

59558 If successful, the *sched_setparam()* function shall return zero.

59559 If the call to *sched_setparam()* is unsuccessful, the priority shall remain unchanged, and the
 59560 function shall return a value of -1 and set *errno* to indicate the error.

59561 **ERRORS**

59562 The *sched_setparam()* function shall fail if:

59563 [EINVAL] One or more of the requested scheduling parameters is outside the range
 59564 defined for the scheduling policy of the specified *pid*.

59565 [EPERM] The requesting process does not have permission to set the scheduling
 59566 parameters for the specified process, or does not have appropriate privileges
 59567 to invoke *sched_setparam()*.

59568 [ESRCH] No process can be found corresponding to that specified by *pid*.

59569 EXAMPLES

59570 None.

59571 APPLICATION USAGE

59572 None.

59573 RATIONALE

59574 None.

59575 FUTURE DIRECTIONS

59576 None.

59577 SEE ALSO

59578 [Scheduling Policies](#) (on page 506), *sched_getparam()*, *sched_getscheduler()*, *sched_setscheduler()*

59579 XBD [<sched.h>](#)

59580 CHANGE HISTORY

59581 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59582 Issue 6

59583 The *sched_setparam()* function is marked as part of the Process Scheduling option.

59584 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59585 implementation does not support the Process Scheduling option.

59586 The following new requirements on POSIX implementations derive from alignment with the
 59587 Single UNIX Specification:

59588 In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
 59589 added.

59590 Sections describing two-level scheduling and atomicity of the function are added.

59591 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

59592 IEEE PASC Interpretation 1003.1 #100 is applied.

59593 Issue 7

59594 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

59595 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
 59596 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

59597 **NAME**59598 sched_setscheduler ‡set scheduling policy and parameters **REALTIME**)59599 **SYNOPSIS**

```
59600 PS #include <sched.h>
59601 int sched_setscheduler(pid_t pid, int policy,
59602     const struct sched_param *param);
```

59603 **DESCRIPTION**

59604 The *sched_setscheduler()* function shall set the scheduling policy and scheduling parameters of
 59605 the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure
 59606 pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param**
 59607 structure shall be any integer within the inclusive priority range for the scheduling policy
 59608 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler()*
 59609 function is unspecified.

59610 The possible values for the *policy* parameter are defined in the **<sched.h>** header.

59611 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 59612 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

59613 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling
 59614 process.

59615 The conditions under which one process has appropriate privileges to change the scheduling
 59616 parameters of another process are implementation-defined.

59617 Implementations may require that the requesting process have permission to set its own
 59618 scheduling parameters or those of another process. Additionally, implementation-defined
 59619 restrictions may apply as to the appropriate privileges required to set the scheduling policy of
 59620 the process, or the scheduling policy of another process, to a particular value.

59621 The *sched_setscheduler()* function shall be considered successful if it succeeds in setting the
 59622 scheduling policy and scheduling parameters of the process specified by *pid* to the values
 59623 specified by *policy* and the structure pointed to by *param*, respectively.

59624 See [Scheduling Policies](#) (on page 506) for a description on how this function affects the
 59625 scheduling of the threads within the target process.

59626 SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or
 59627 SCHED_SPORADIC, the result is implementation-defined; this case includes the
 59628 SCHED_OTHER policy.

59629 SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified
 59630 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

59631 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
 59632 function to succeed; if not, the function shall fail. It is unspecified whether the
 59633 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
 59634 rounded to align with the resolution of the clock being used.

59635 This function is not atomic with respect to other threads in the process. Threads may continue to
 59636 execute while this function call is in the process of changing the scheduling policy and
 59637 associated scheduling parameters for the underlying kernel-scheduled entities used by the
 59638 process contention scope threads.

59639 **RETURN VALUE**

59640 Upon successful completion, the function shall return the former scheduling policy of the
 59641 specified process. If the *sched_setscheduler()* function fails to complete successfully, the policy
 59642 and scheduling parameters shall remain unchanged, and the function shall return a value of -1
 59643 and set *errno* to indicate the error.

59644 **ERRORS**

59645 The *sched_setscheduler()* function shall fail if:

59646 [EINVAL] The value of the *policy* parameter is invalid, or one or more of the parameters
 59647 contained in *param* is outside the valid range for the specified scheduling
 59648 policy.

59649 [EPERM] The requesting process does not have permission to set either or both of the
 59650 scheduling parameters or the scheduling policy of the specified process.

59651 [ESRCH] No process can be found corresponding to that specified by *pid*.

59652 **EXAMPLES**

59653 None.

59654 **APPLICATION USAGE**

59655 None.

59656 **RATIONALE**

59657 None.

59658 **FUTURE DIRECTIONS**

59659 None.

59660 **SEE ALSO**

59661 [Scheduling Policies](#) (on page 506), [sched_getparam\(\)](#), [sched_getscheduler\(\)](#), [sched_setparam\(\)](#)

59662 XBD [<sched.h>](#)

59663 **CHANGE HISTORY**

59664 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59665 **Issue 6**

59666 The *sched_setscheduler()* function is marked as part of the Process Scheduling option.

59667 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59668 implementation does not support the Process Scheduling option.

59669 The following new requirements on POSIX implementations derive from alignment with the
 59670 Single UNIX Specification:

59671 In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
 59672 added.

59673 Sections describing two-level scheduling and atomicity of the function are added.

59674 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

59675 **Issue 7**

59676 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

59677 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements
 59678 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

59679 **NAME**

59680 sched_yield — yield the processor

59681 **SYNOPSIS**

59682 #include <sched.h>

59683 int sched_yield(void);

59684 **DESCRIPTION**59685 The *sched_yield()* function shall force the running thread to relinquish the processor until it again
59686 becomes the head of its thread list. It takes no arguments.59687 **RETURN VALUE**59688 The *sched_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a
59689 value of -1 and set *errno* to indicate the error.59690 **ERRORS**

59691 No errors are defined.

59692 **EXAMPLES**

59693 None.

59694 **APPLICATION USAGE**59695 The conceptual model for scheduling semantics in POSIX.1-2017 defines a set of thread lists. This
59696 set of thread lists is always present regardless of the scheduling options supported by the
59697 system. On a system where the Process Scheduling option is not supported, portable
59698 applications should not make any assumptions regarding whether threads from other processes
59699 will be on the same thread list.59700 **RATIONALE**

59701 None.

59702 **FUTURE DIRECTIONS**

59703 None.

59704 **SEE ALSO**59705 XBD [<sched.h>](#)59706 **CHANGE HISTORY**59707 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
59708 POSIX Threads Extension.59709 **Issue 6**59710 The *sched_yield()* function is now marked as part of the Process Scheduling and Threads options.59711 **Issue 7**

59712 SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

59713 The *sched_yield()* function is moved to the Base.

59714 **NAME**

59715 seed48 ‡seed a uniformly distributed pseudo-random non-negative long integer generator

59716 **SYNOPSIS**

59717 XSI #include <stdlib.h>

59718 unsigned short *seed48(unsigned short seed16v[3]);

59719 **DESCRIPTION**59720 Refer to *drand48()*.

59721 **NAME**

59722 seekdir — set the position of a directory stream

59723 **SYNOPSIS**

```
59724 XSI #include <dirent.h>  
59725 void seekdir(DIR *dirp, long loc);
```

59726 **DESCRIPTION**

59727 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory
59728 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been
59729 returned from an earlier call to *telldir()* using the same directory stream. The new position
59730 reverts to the one associated with the directory stream when *telldir()* was performed.

59731 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*
59732 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to
59733 *readdir()* are unspecified.

59734 **RETURN VALUE**59735 The *seekdir()* function shall not return a value.59736 **ERRORS**

59737 No errors are defined.

59738 **EXAMPLES**

59739 None.

59740 **APPLICATION USAGE**

59741 None.

59742 **RATIONALE**

59743 The original standard developers perceived that there were restrictions on the use of the
59744 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these
59745 functions need not be supported on all POSIX-conforming systems. They are required on
59746 implementations supporting the XSI option.

59747 One of the perceived problems of implementation is that returning to a given point in a directory
59748 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-
59749 trees, hashing functions, or other similar mechanisms to order their directories are considered.
59750 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a
59751 given directory entry will be seen at all, or more than once.

59752 On systems not supporting these functions, their capability can sometimes be accomplished by
59753 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to
59754 relocate the position from which the filename was saved.

59755 **FUTURE DIRECTIONS**

59756 None.

59757 **SEE ALSO**59758 *fdopendir()*, *readdir()*, *telldir()*

59759 XBD <dirent.h>, <sys/types.h>

59760 **CHANGE HISTORY**

59761 First released in Issue 2.

59762 **Issue 6**

59763 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

59764 **Issue 7**

59765 SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should
59766 have been returned from an earlier call to *telldir()* using the same directory stream.

59767 **NAME**

59768 select — synchronous I/O multiplexing

59769 **SYNOPSIS**

```
59770     #include <sys/select.h>
59771     int select(int nfds, fd_set *restrict readfds,
59772               fd_set *restrict writefds, fd_set *restrict errorfds,
59773               struct timeval *restrict timeout);
```

59774 **DESCRIPTION**59775 Refer to [pselect\(\)](#).

59776 **NAME**

59777 sem_close — close a named semaphore

59778 **SYNOPSIS**

59779 #include <semaphore.h>

59780 int sem_close(sem_t *sem);

59781 **DESCRIPTION**

59782 The *sem_close()* function shall indicate that the calling process is finished using the named
59783 semaphore indicated by *sem*. The effects of calling *sem_close()* for an unnamed semaphore (one
59784 created by *sem_init()*) are undefined. The *sem_close()* function shall deallocate (that is, make
59785 available for reuse by a subsequent *sem_open()* by this process) any system resources allocated
59786 by the system for use by this process for this semaphore. The effect of subsequent use of the
59787 semaphore indicated by *sem* by this process is undefined. If any threads in the calling process are
59788 currently blocked on the semaphore, the behavior is undefined. If the semaphore has not been
59789 removed with a successful call to *sem_unlink()*, then *sem_close()* has no effect on the state of the
59790 semaphore. If the *sem_unlink()* function has been successfully invoked for *name* after the most
59791 recent call to *sem_open()* with O_CREAT for this semaphore, then when all processes that have
59792 opened the semaphore close it, the semaphore is no longer accessible.

59793 **RETURN VALUE**

59794 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be
59795 returned and *errno* set to indicate the error.

59796 **ERRORS**59797 The *sem_close()* function may fail if:59798 [EINVAL] The *sem* argument is not a valid semaphore descriptor.59799 **EXAMPLES**

59800 None.

59801 **APPLICATION USAGE**

59802 None.

59803 **RATIONALE**

59804 None.

59805 **FUTURE DIRECTIONS**

59806 None.

59807 **SEE ALSO**59808 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#), [sem_unlink\(\)](#)59809 XBD [<semaphore.h>](#)59810 **CHANGE HISTORY**

59811 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59812 **Issue 6**59813 The *sem_close()* function is marked as part of the Semaphores option.

59814 The [ENOSYS] error condition has been removed as stubs need not be provided if an
59815 implementation does not support the Semaphores option.

59816 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/113 is applied, updating the ERRORS
59817 section so that the [EINVAL] error becomes optional.

59818 **Issue 7**

59819 The *sem_close()* function is moved from the Semaphores option to the Base.

59820 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0523 [37] is applied.

59821 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0317 [870] is applied.

59822 **NAME**

59823 sem_destroy — destroy an unnamed semaphore

59824 **SYNOPSIS**

59825 #include <semaphore.h>

59826 int sem_destroy(sem_t *sem);

59827 **DESCRIPTION**

59828 The *sem_destroy()* function shall destroy the unnamed semaphore indicated by *sem*. Only a
59829 semaphore that was created using *sem_init()* may be destroyed using *sem_destroy()*; the effect of
59830 calling *sem_destroy()* with a named semaphore is undefined. The effect of subsequent use of the
59831 semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem_init()*.

59832 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The
59833 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

59834 **RETURN VALUE**

59835 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be
59836 returned and *errno* set to indicate the error.

59837 **ERRORS**59838 The *sem_destroy()* function may fail if:59839 [EINVAL] The *sem* argument is not a valid semaphore.

59840 [EBUSY] There are currently processes blocked on the semaphore.

59841 **EXAMPLES**

59842 None.

59843 **APPLICATION USAGE**

59844 None.

59845 **RATIONALE**

59846 None.

59847 **FUTURE DIRECTIONS**

59848 None.

59849 **SEE ALSO**59850 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#)59851 XBD [<semaphore.h>](#)59852 **CHANGE HISTORY**

59853 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59854 **Issue 6**59855 The *sem_destroy()* function is marked as part of the Semaphores option.

59856 The [ENOSYS] error condition has been removed as stubs need not be provided if an
59857 implementation does not support the Semaphores option.

59858 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/114 is applied, updating the ERRORS
59859 section so that the [EINVAL] error becomes optional.

59860 **Issue 7**59861 The *sem_destroy()* function is moved from the Semaphores option to the Base.

59862 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0524 [37] is applied.

59863 **NAME**

59864 sem_getvalue — get the value of a semaphore

59865 **SYNOPSIS**

59866 #include <semaphore.h>

59867 int sem_getvalue(sem_t *restrict sem, int *restrict sval);

59868 **DESCRIPTION**

59869 The *sem_getvalue()* function shall update the location referenced by the *sval* argument to have
 59870 the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The
 59871 updated value represents an actual semaphore value that occurred at some unspecified time
 59872 during the call, but it need not be the actual value of the semaphore when it is returned to the
 59873 calling process.

59874 If *sem* is locked, then the object to which *sval* points shall either be set to zero or to a negative
 59875 number whose absolute value represents the number of processes waiting for the semaphore at
 59876 some unspecified time during the call.

59877 **RETURN VALUE**

59878 Upon successful completion, the *sem_getvalue()* function shall return a value of zero. Otherwise,
 59879 it shall return a value of -1 and set *errno* to indicate the error.

59880 **ERRORS**59881 The *sem_getvalue()* function may fail if:59882 [EINVAL] The *sem* argument does not refer to a valid semaphore.59883 **EXAMPLES**

59884 None.

59885 **APPLICATION USAGE**

59886 None.

59887 **RATIONALE**

59888 None.

59889 **FUTURE DIRECTIONS**

59890 None.

59891 **SEE ALSO**59892 *semctl()*, *semget()*, *semop()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*59893 XBD <[semaphore.h](#)>59894 **CHANGE HISTORY**

59895 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59896 **Issue 6**59897 The *sem_getvalue()* function is marked as part of the Semaphores option.

59898 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 59899 implementation does not support the Semaphores option.

59900 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std
 59901 1003.1d-1999.

59902 The **restrict** keyword is added to the *sem_getvalue()* prototype for alignment with the
 59903 ISO/IEC 9899:1999 standard.

59904 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

59905 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/115 is applied, updating the ERRORS
59906 section so that the [EINVAL] error becomes optional.

59907 **Issue 7**

59908 The *sem_getvalue()* function is moved from the Semaphores option to the Base.

59909 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0525 [37] is applied.

59910 **NAME**

59911 sem_init — initialize an unnamed semaphore

59912 **SYNOPSIS**

59913 #include <semaphore.h>

59914 int sem_init(sem_t *sem, int pshared, unsigned value);

59915 **DESCRIPTION**

59916 The `sem_init()` function shall initialize the unnamed semaphore referred to by `sem`. The value of
 59917 the initialized semaphore shall be `value`. Following a successful call to `sem_init()`, the semaphore
 59918 may be used in subsequent calls to `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and
 59919 `sem_destroy()`. This semaphore shall remain usable until the semaphore is destroyed.

59920 If the `pshared` argument has a non-zero value, then the semaphore is shared between processes;
 59921 in this case, any process that can access the semaphore `sem` can use `sem` for performing
 59922 `sem_wait()`, `sem_timedwait()`, `sem_trywait()`, `sem_post()`, and `sem_destroy()` operations.

59923 If the `pshared` argument is zero, then the semaphore is shared between threads of the process; any
 59924 thread in this process can use `sem` for performing `sem_wait()`, `sem_timedwait()`, `sem_trywait()`,
 59925 `sem_post()`, and `sem_destroy()` operations.

59926 See [Section 2.9.9](#) (on page 523) for further requirements.

59927 Attempting to initialize an already initialized semaphore results in undefined behavior.

59928 **RETURN VALUE**

59929 Upon successful completion, the `sem_init()` function shall initialize the semaphore in `sem` and
 59930 return 0. Otherwise, it shall return `-1` and set `errno` to indicate the error.

59931 **ERRORS**59932 The `sem_init()` function shall fail if:59933 [EINVAL] The `value` argument exceeds {SEM_VALUE_MAX}.

59934 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the
 59935 limit on semaphores ({SEM_NSEMS_MAX}) has been reached.

59936 [EPERM] The process lacks appropriate privileges to initialize the semaphore.

59937 **EXAMPLES**

59938 None.

59939 **APPLICATION USAGE**

59940 None.

59941 **RATIONALE**

59942 None.

59943 **FUTURE DIRECTIONS**

59944 None.

59945 **SEE ALSO**59946 [sem_destroy\(\)](#), [sem_post\(\)](#), [sem_timedwait\(\)](#), [sem_trywait\(\)](#)59947 XBD [<semaphore.h>](#)59948 **CHANGE HISTORY**

59949 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59950 **Issue 6**

59951 The *sem_init()* function is marked as part of the Semaphores option.

59952 The [ENOSYS] error condition has been removed as stubs need not be provided if an
59953 implementation does not support the Semaphores option.

59954 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std
59955 1003.1d-1999.

59956 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/116 is applied, updating the
59957 DESCRIPTION to add the *sem_timedwait()* function for alignment with IEEE Std 1003.1d-1999.

59958 **Issue 7**

59959 SD5-XSH-ERN-176 is applied.

59960 The *sem_init()* function is moved from the Semaphores option to the Base.

59961 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0526 [37] is applied.

59962 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0318 [972] is applied.

59963 **NAME**

59964 sem_open — initialize and open a named semaphore

59965 **SYNOPSIS**

59966 #include <semaphore.h>

59967 sem_t *sem_open(const char *name, int oflag, ...);

59968 **DESCRIPTION**

59969 The *sem_open()* function shall establish a connection between a named semaphore and a process.
 59970 Following a call to *sem_open()* with semaphore name *name*, the process may reference the
 59971 semaphore associated with *name* using the address returned from the call. This semaphore may
 59972 be used in subsequent calls to *sem_wait()*, *sem_timedwait()*, *sem_trywait()*, *sem_post()*, and
 59973 *sem_close()*. The semaphore remains usable by this process until the semaphore is closed by a
 59974 successful call to *sem_close()*, *_exit()*, or one of the *exec* functions.

59975 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to
 59976 *sem_open()*. The following flag bits may be set in *oflag*:

59977 **O_CREAT** This flag is used to create a semaphore if it does not already exist. If **O_CREAT** is
 59978 set and the semaphore already exists, then **O_CREAT** has no effect, except as noted
 59979 under **O_EXCL**. Otherwise, *sem_open()* creates a named semaphore. The **O_CREAT**
 59980 flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and
 59981 *value*, which is of type **unsigned**. The semaphore is created with an initial value of
 59982 *value*. Valid initial values for semaphores are less than or equal to
 59983 {SEM_VALUE_MAX}.

59984 The user ID of the semaphore shall be set to the effective user ID of the process.
 59985 The group ID of the semaphore shall be set to the effective group ID of the process;
 59986 however, if the *name* argument is visible in the file system, the group ID may be set
 59987 to the group ID of the containing directory. The permission bits of the semaphore
 59988 are set to the value of the *mode* argument except those set in the file mode creation
 59989 mask of the process. When bits in *mode* other than the file permission bits are
 59990 specified, the effect is unspecified.

59991 After the semaphore named *name* has been created by *sem_open()* with the
 59992 **O_CREAT** flag, other processes can connect to the semaphore by calling
 59993 *sem_open()* with the same value of *name*.

59994 **O_EXCL** If **O_EXCL** and **O_CREAT** are set, *sem_open()* fails if the semaphore *name* exists.
 59995 The check for the existence of the semaphore and the creation of the semaphore if it
 59996 does not exist are atomic with respect to other processes executing *sem_open()* with
 59997 **O_EXCL** and **O_CREAT** set. If **O_EXCL** is set and **O_CREAT** is not set, the effect is
 59998 undefined.

59999 If flags other than **O_CREAT** and **O_EXCL** are specified in the *oflag* parameter, the
 60000 effect is unspecified.

60001 The *name* argument points to a string naming a semaphore object. It is unspecified whether the
 60002 name appears in the file system and is visible to functions that take pathnames as arguments.
 60003 The *name* argument conforms to the construction rules for a pathname, except that the
 60004 interpretation of <slash> characters other than the leading <slash> character in *name* is
 60005 implementation-defined, and that the length limits for the *name* argument are implementation-
 60006 defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If
 60007 *name* begins with the <slash> character, then processes calling *sem_open()* with the same value of
 60008 *name* shall refer to the same semaphore object, as long as that name has not been removed. If
 60009 *name* does not begin with the <slash> character, the effect is implementation-defined.

60010 If a process makes multiple successful calls to *sem_open()* with the same value for *name*, the same
 60011 semaphore address shall be returned for each such successful call, provided that there have been
 60012 no calls to *sem_unlink()* for this semaphore, and at least one previous successful *sem_open()* call
 60013 for this semaphore has not been matched with a *sem_close()* call.

60014 References to copies of the semaphore produce undefined results.

60015 RETURN VALUE

60016 Upon successful completion, the *sem_open()* function shall return the address of the semaphore.
 60017 Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol
 60018 SEM_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem_open()*
 60019 shall return the value SEM_FAILED.

60020 ERRORS

60021 If any of the following conditions occur, the *sem_open()* function shall return SEM_FAILED and
 60022 set *errno* to the corresponding value:

60023 [EACCES] The named semaphore exists and the permissions specified by *oflag* are
 60024 denied, or the named semaphore does not exist and permission to create the
 60025 named semaphore is denied.

60026 [EEXIST] O_CREAT and O_EXCL are set and the named semaphore already exists.

60027 [EINTR] The *sem_open()* operation was interrupted by a signal.

60028 [EINVAL] The *sem_open()* operation is not supported for the given name, or O_CREAT
 60029 was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.

60030 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this
 60031 process.

60032 [ENFILE] Too many semaphores are currently open in the system.

60033 [ENOENT] O_CREAT is not set and the named semaphore does not exist.

60034 [ENOMEM] There is insufficient memory for the creation of the new named semaphore.

60035 [ENOSPC] There is insufficient space on a storage device for the creation of the new
 60036 named semaphore.

60037 If any of the following conditions occur, the *sem_open()* function may return SEM_FAILED and
 60038 set *errno* to the corresponding value:

60039 [ENAMETOOLONG]

60040 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 60041 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 60042 systems, or has a pathname component that is longer than
 60043 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 60044 longer than {_XOPEN_NAME_MAX} on XSI systems.

60045 EXAMPLES

60046 None.

60047 APPLICATION USAGE

60048 None.

60049 RATIONALE

60050 Early drafts required an error return value of -1 with the type `sem_t *` for the `sem_open()`
60051 function, which is not guaranteed to be portable across implementations. The revised text
60052 provides the symbolic error code `SEM_FAILED` to eliminate the type conflict.

60053 FUTURE DIRECTIONS

60054 A future version might require the `sem_open()` and `sem_unlink()` functions to have semantics
60055 similar to normal file system operations.

60056 SEE ALSO

60057 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem_close\(\)](#), [sem_post\(\)](#), [sem_timedwait\(\)](#), [sem_trywait\(\)](#), [sem_unlink\(\)](#)

60058 XBD [<semaphore.h>](#)

60059 CHANGE HISTORY

60060 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60061 Issue 6

60062 The `sem_open()` function is marked as part of the Semaphores option.

60063 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
60064 implementation does not support the Semaphores option.

60065 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with IEEE Std
60066 1003.1d-1999.

60067 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/117 is applied, updating the
60068 DESCRIPTION to add the `sem_timedwait()` function for alignment with IEEE Std 1003.1d-1999.

60069 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/118 is applied, updating the
60070 DESCRIPTION to describe the conditions to return the same semaphore address on a call to
60071 `sem_open()`. The words “and at least one previous successful `sem_open()` call for this semaphore
60072 has not been matched with a `sem_close()` call” are added.

60073 Issue 7

60074 Austin Group Interpretation 1003.1-2001 #066 is applied, updating the `[ENOSPC]` error case and
60075 adding the `[ENOMEM]` error case.

60076 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the `name` argument and
60077 adding `[ENAMETOOLONG]` as a “may fail” error.

60078 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

60079 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the
60080 user ID and group ID of the semaphore.

60081 The `sem_open()` function is moved from the Semaphores option to the Base.

60082 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0527 [37] is applied.

60083 **NAME**

60084 sem_post — unlock a semaphore

60085 **SYNOPSIS**

60086 #include <semaphore.h>

60087 int sem_post(sem_t *sem);

60088 **DESCRIPTION**60089 The *sem_post()* function shall unlock the semaphore referenced by *sem* by performing a
60090 semaphore unlock operation on that semaphore.60091 If the semaphore value resulting from this operation is positive, then no threads were blocked
60092 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.60093 If the value of the semaphore resulting from this operation is zero, then one of the threads
60094 blocked waiting for the semaphore shall be allowed to return successfully from its call to
60095 *sem_wait()*. If the Process Scheduling option is supported, the thread to be unblocked shall be
60096 chosen in a manner appropriate to the scheduling policies and parameters in effect for the
60097 blocked threads. In the case of the schedulers SCHED_FIFO and SCHED_RR, the highest
60098 priority waiting thread shall be unblocked, and if there is more than one highest priority thread
60099 blocked waiting for the semaphore, then the highest priority thread that has been waiting the
60100 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread
60101 to unblock is unspecified.60102 SS If the Process Sporadic Server option is supported, and the scheduling policy is
60103 SCHED_SPORADIC, the semantics are as per SCHED_FIFO above.60104 The *sem_post()* function shall be async-signal-safe and may be invoked from a signal-catching
60105 function.60106 **RETURN VALUE**60107 If successful, the *sem_post()* function shall return zero; otherwise, the function shall return -1
60108 and set *errno* to indicate the error.60109 **ERRORS**60110 The *sem_post()* function may fail if:60111 [EINVAL] The *sem* argument does not refer to a valid semaphore.60112 **EXAMPLES**60113 See *sem_timedwait()*.60114 **APPLICATION USAGE**

60115 None.

60116 **RATIONALE**

60117 None.

60118 **FUTURE DIRECTIONS**

60119 None.

60120 **SEE ALSO**60121 *semctl()*, *semget()*, *semop()*, *sem_timedwait()*, *sem_trywait()*

60122 XBD Section 4.12 (on page 111), <semaphore.h>

60123 **CHANGE HISTORY**

60124 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60125 **Issue 6**

60126 The *sem_post()* function is marked as part of the Semaphores option.

60127 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60128 implementation does not support the Semaphores option.

60129 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std
60130 1003.1d-1999.

60131 SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be
60132 unblocked is specified for alignment with IEEE Std 1003.1d-1999.

60133 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/119 is applied, updating the ERRORS
60134 section so that the [EINVAL] error becomes optional.

60135 **Issue 7**

60136 Austin Group Interpretation 1003.1-2001 #156 is applied.

60137 The *sem_post()* function is moved from the Semaphores option to the Base.

60138 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0528 [37] is applied.

60139 **NAME**

60140 sem_timedwait — lock a semaphore

60141 **SYNOPSIS**

60142 #include <semaphore.h>

60143 #include <time.h>

```
60144 int sem_timedwait(sem_t *restrict sem,
60145                  const struct timespec *restrict abstime);
```

60146 **DESCRIPTION**

60147 The *sem_timedwait()* function shall lock the semaphore referenced by *sem* as in the *sem_wait()*
 60148 function. However, if the semaphore cannot be locked without waiting for another process or
 60149 thread to unlock the semaphore by performing a *sem_post()* function, this wait shall be
 60150 terminated when the specified timeout expires.

60151 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
 60152 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 60153 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
 60154 call.

60155 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 60156 be the resolution of the clock on which it is based. The **timespec** data type is defined as a
 60157 structure in the **<time.h>** header.

60158 Under no circumstance shall the function fail with a timeout if the semaphore can be locked
 60159 immediately. The validity of the *abstime* need not be checked if the semaphore can be locked
 60160 immediately.

60161 **RETURN VALUE**

60162 The *sem_timedwait()* function shall return zero if the calling process successfully performed the
 60163 semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the
 60164 state of the semaphore shall be unchanged, and the function shall return a value of -1 and set
 60165 *errno* to indicate the error.

60166 **ERRORS**60167 The *sem_timedwait()* function shall fail if:

60168 [EINVAL] The process or thread would have blocked, and the *abstime* parameter
 60169 specified a nanoseconds field value less than zero or greater than or equal to
 60170 1 000 million.

60171 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

60172 The *sem_timedwait()* function may fail if:

60173 [EDEADLK] A deadlock condition was detected.

60174 [EINTR] A signal interrupted this function.

60175 [EINVAL] The *sem* argument does not refer to a valid semaphore.

60176 **EXAMPLES**

60177 The program shown below operates on an unnamed semaphore. The program expects two
 60178 command-line arguments. The first argument specifies a seconds value that is used to set an
 60179 alarm timer to generate a SIGALRM signal. This handler performs a *sem_post(3)* to increment the
 60180 semaphore that is being waited on in *main()* using *sem_timedwait()*. The second command-line
 60181 argument specifies the length of the timeout, in seconds, for *sem_timedwait()*.

```

60182 #include <unistd.h>
60183 #include <stdio.h>
60184 #include <stdlib.h>
60185 #include <semaphore.h>
60186 #include <time.h>
60187 #include <assert.h>
60188 #include <errno.h>
60189 #include <signal.h>

60190 sem_t sem;

60191 static void
60192 handler(int sig)
60193 {
60194     int sav_errno = errno;
60195     static const char info_msg[] = "sem_post() from handler\n";
60196     write(STDOUT_FILENO, info_msg, sizeof info_msg - 1);
60197     if (sem_post(&sem) == -1) {
60198         static const char err_msg[] = "sem_post() failed\n";
60199         write(STDERR_FILENO, err_msg, sizeof err_msg - 1);
60200         _exit(EXIT_FAILURE);
60201     }
60202     errno = sav_errno;
60203 }

60204 int
60205 main(int argc, char *argv[])
60206 {
60207     struct sigaction sa;
60208     struct timespec ts;
60209     int s;

60210     if (argc != 3) {
60211         fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
60212             argv[0]);
60213         exit(EXIT_FAILURE);
60214     }

60215     if (sem_init(&sem, 0, 0) == -1) {
60216         perror("sem_init");
60217         exit(EXIT_FAILURE);
60218     }

60219     /* Establish SIGALRM handler; set alarm timer using argv[1] */

60220     sa.sa_handler = handler;
60221     sigemptyset(&sa.sa_mask);
60222     sa.sa_flags = 0;
60223     if (sigaction(SIGALRM, &sa, NULL) == -1) {

```

```

60224         perror("sigaction");
60225         exit(EXIT_FAILURE);
60226     }
60227     alarm(atoi(argv[1]));
60228     /* Calculate relative interval as current time plus
60229        number of seconds given argv[2] */
60230     if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
60231         perror("clock_gettime");
60232         exit(EXIT_FAILURE);
60233     }
60234     ts.tv_sec += atoi(argv[2]);
60235     printf("main() about to call sem_timedwait()\n");
60236     while ((s = sem_timedwait(&sem, &ts)) == -1 && errno == EINTR)
60237         continue; /* Restart if interrupted by handler */
60238     /* Check what happened */
60239     if (s == -1) {
60240         if (errno == ETIMEDOUT)
60241             printf("sem_timedwait() timed out\n");
60242         else
60243             perror("sem_timedwait");
60244     } else
60245         printf("sem_timedwait() succeeded\n");
60246     exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
60247 }

```

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD [Section 3.291](#) (on page 80).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[sem_post\(\)](#), [sem_trywait\(\)](#), [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [time\(\)](#)

XBD [Section 3.291](#) (on page 80), [<semaphore.h>](#), [<time.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/120 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The `sem_timedwait()` function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

An example is added.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0529 [138] is applied.

60267 **NAME**

60268 sem_trywait, sem_wait — lock a semaphore

60269 **SYNOPSIS**

```
60270 #include <semaphore.h>
60271 int sem_trywait(sem_t *sem);
60272 int sem_wait(sem_t *sem);
```

60273 **DESCRIPTION**

60274 The *sem_trywait()* function shall lock the semaphore referenced by *sem* only if the semaphore is
 60275 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not
 60276 lock the semaphore.

60277 The *sem_wait()* function shall lock the semaphore referenced by *sem* by performing a semaphore
 60278 lock operation on that semaphore. If the semaphore value is currently zero, then the calling
 60279 thread shall not return from the call to *sem_wait()* until it either locks the semaphore or the call is
 60280 interrupted by a signal.

60281 Upon successful return, the state of the semaphore shall be locked and shall remain locked until
 60282 the *sem_post()* function is executed and returns successfully.

60283 The *sem_wait()* function is interruptible by the delivery of a signal.

60284 **RETURN VALUE**

60285 The *sem_trywait()* and *sem_wait()* functions shall return zero if the calling process successfully
 60286 performed the semaphore lock operation on the semaphore designated by *sem*. If the call was
 60287 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a
 60288 value of -1 and set *errno* to indicate the error.

60289 **ERRORS**

60290 The *sem_trywait()* function shall fail if:

60291 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the
 60292 *sem_trywait()* operation.

60293 The *sem_trywait()* and *sem_wait()* functions may fail if:

60294 [EDEADLK] A deadlock condition was detected.

60295 [EINTR] A signal interrupted this function.

60296 [EINVAL] The *sem* argument does not refer to a valid semaphore.

60297 **EXAMPLES**

60298 None.

60299 **APPLICATION USAGE**

60300 Applications using these functions may be subject to priority inversion, as discussed in XBD
 60301 [Section 3.291](#) (on page 80).

60302 **RATIONALE**

60303 None.

60304 **FUTURE DIRECTIONS**

60305 None.

60306 **SEE ALSO**

60307 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem_post\(\)](#), [sem_timedwait\(\)](#)

60308 XBD [Section 3.291](#) (on page 80), [Section 4.12](#) (on page 111), [<semaphore.h>](#)

60309 **CHANGE HISTORY**

60310 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60311 **Issue 6**

60312 The *sem_trywait()* and *sem_wait()* functions are marked as part of the Semaphores option.

60313 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60314 implementation does not support the Semaphores option.

60315 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std
60316 1003.1d-1999.

60317 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/121 is applied, updating the ERRORS
60318 section so that the [EINVAL] error becomes optional.

60319 **Issue 7**

60320 SD5-XSH-ERN-54 is applied, removing the *sem_wait()* function from the “shall fail” error cases.

60321 The *sem_trywait()* and *sem_wait()* functions are moved from the Semaphores option to the Base.
60322 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0530 [37] is applied.

60323 **NAME**

60324 sem_unlink — remove a named semaphore

60325 **SYNOPSIS**

60326 #include <semaphore.h>

60327 int sem_unlink(const char *name);

60328 **DESCRIPTION**

60329 The *sem_unlink()* function shall remove the semaphore named by the string *name*. If the
 60330 semaphore named by *name* is currently referenced by other processes, then *sem_unlink()* shall
 60331 have no effect on the state of the semaphore. If one or more processes have the semaphore open
 60332 when *sem_unlink()* is called, destruction of the semaphore is postponed until all references to the
 60333 semaphore have been destroyed by calls to *sem_close()*, *_exit()*, or *exec*. Calls to *sem_open()* to
 60334 recreate or reconnect to the semaphore refer to a new semaphore after *sem_unlink()* is called. The
 60335 *sem_unlink()* call shall not block until all references have been destroyed; it shall return
 60336 immediately.

60337 **RETURN VALUE**

60338 Upon successful completion, the *sem_unlink()* function shall return a value of 0. Otherwise, the
 60339 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to
 60340 indicate the error.

60341 **ERRORS**60342 The *sem_unlink()* function shall fail if:

60343 [EACCES] Permission is denied to unlink the named semaphore.

60344 [ENOENT] The named semaphore does not exist.

60345 The *sem_unlink()* function may fail if:

60346 [ENAMETOOLONG]

60347 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems
 60348 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI
 60349 systems, or has a pathname component that is longer than
 60350 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or
 60351 longer than `{_XOPEN_NAME_MAX}` on XSI systems. A call to *sem_unlink()*
 60352 with a *name* argument that contains the same semaphore name as was
 60353 previously used in a successful *sem_open()* call shall not give an
 60354 [ENAMETOOLONG] error.

60355 **EXAMPLES**

60356 None.

60357 **APPLICATION USAGE**

60358 None.

60359 **RATIONALE**

60360 None.

60361 **FUTURE DIRECTIONS**

60362 A future version might require the *sem_open()* and *sem_unlink()* functions to have semantics
 60363 similar to normal file system operations.

60364 **SEE ALSO**60365 *semctl()*, *semget()*, *semop()*, *sem_close()*, *sem_open()*

60366 XBD <semaphore.h>

60367 **CHANGE HISTORY**

60368 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

60369 **Issue 6**

60370 The *sem_unlink()* function is marked as part of the Semaphores option.

60371 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60372 implementation does not support the Semaphores option.

60373 **Issue 7**

60374 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
60375 ``shall fail'' to a ``may fail'' error.

60376 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

60377 The *sem_unlink()* function is moved from the Semaphores option to the Base.

60378 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0531 [37] is applied.

60379 **NAME**

60380 sem_wait — lock a semaphore

60381 **SYNOPSIS**

60382 #include <semaphore.h>

60383 int sem_wait(sem_t *sem);

60384 **DESCRIPTION**

60385 Refer to *sem_trywait()*.

60386 **NAME**

60387 semctl ‡XSI semaphore control operations

60388 **SYNOPSIS**

```
60389 XSI #include <sys/sem.h>
60390 int semctl(int semid, int semnum, int cmd, ...);
```

60391 **DESCRIPTION**

60392 The *semctl()* function operates on XSI semaphores (see XBD [Section 4.17](#), on page 114). It is
 60393 unspecified whether this function interoperates with the realtime interprocess communication
 60394 facilities defined in [Section 2.8](#) (on page 503).

60395 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.
 60396 The fourth argument is optional and depends upon the operation requested. If required, it is of
 60397 type **union semun**, which the application shall explicitly declare:

```
60398 union semun {
60399     int val;
60400     struct semid_ds *buf;
60401     unsigned short *array;
60402 } arg;
```

60403 Each operation shall be performed atomically.

60404 The following semaphore control operations as specified by *cmd* are executed with respect to the
 60405 semaphore specified by *semid* and *semnum*. The level of permission required for each operation
 60406 is shown with each command; see [Section 2.7](#) (on page 501). The symbolic names for the values
 60407 of *cmd* are defined in the `<sys/sem.h>` header:

60408	GETVAL	Return the value of <i>semval</i> ; see <code><sys/sem.h></code> . Requires read permission.
60409	SETVAL	Set the value of <i>semval</i> to <i>arg.val</i> , where <i>arg</i> is the value of the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. Also, the <i>sem_ctime</i> timestamp shall be set to the current time, as described in Section 2.7.1 (on page 502). Requires alter permission; see Section 2.7 (on page 501).
60410		
60411		
60412		
60413		
60414	GETPID	Return the value of <i>sempid</i> . Requires read permission.
60415	GETNCNT	Return the value of <i>semmcnt</i> . Requires read permission.
60416	GETZCNT	Return the value of <i>semzcnt</i> . Requires read permission.

60417 The following values of *cmd* operate on each *semval* in the set of semaphores:

60418	GETALL	Return the value of <i>semval</i> for each semaphore in the semaphore set and place into the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . Requires read permission.
60419		
60420		
60421	SETALL	Set the value of <i>semval</i> for each semaphore in the semaphore set according to the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. Also, the <i>sem_ctime</i> timestamp shall be set to the current time, as described in Section 2.7.1 (on page 502). Requires alter permission.
60422		
60423		
60424		
60425		
60426		

60427		The following values of <i>cmd</i> are also available:
60428	IPC_STAT	Place the current value of each member of the semid_ds data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . The contents of this structure are defined in <sys/sem.h> . Requires read permission.
60429		
60430		
60431		
60432	IPC_SET	Set the value of the following members of the semid_ds data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> :
60433		
60434		
60435		<code>sem_perm.uid</code>
60436		<code>sem_perm.gid</code>
60437		<code>sem_perm.mode</code>
60438		The mode bits specified in Section 2.7.1 (on page 502) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i> . The stored values of any other bits are unspecified. The <i>sem_ctime</i> timestamp shall be set to the current time, as described in Section 2.7.1 (on page 502).
60439		
60440		
60441		
60442		This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i> .
60443		
60444		
60445		
60446	IPC_RMID	Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i> .
60447		
60448		
60449		
60450		
60451		
60452	RETURN VALUE	
60453		If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:
60454	GETVAL	The value of <i>semval</i> .
60455	GETPID	The value of <i>sempid</i> .
60456	GETNCNT	The value of <i>semmcnt</i> .
60457	GETZCNT	The value of <i>semzcnt</i> .
60458	All others	0.
60459		Otherwise, <i>semctl()</i> shall return <code>-1</code> and set <i>errno</i> to indicate the error.
60460	ERRORS	
60461		The <i>semctl()</i> function shall fail if:
60462	[EACCES]	Operation permission is denied to the calling process; see Section 2.7 (on page 501).
60463		
60464	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i> , or the value of <i>cmd</i> is not a valid command.
60465		
60466		
60467	[EPERM]	The argument <i>cmd</i> is equal to <code>IPC_RMID</code> or <code>IPC_SET</code> and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in
60468		
60469		

60470 the data structure associated with *semid*.
60471 [ERANGE] The argument *cmd* is equal to SETVAL or SETALL and the value to which
60472 *semval* is to be set is greater than the system-imposed maximum.

60473 EXAMPLES

60474 Refer to *semop()*.

60475 APPLICATION USAGE

60476 The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a
60477 clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for
60478 backwards-compatibility.

60479 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
60480 Application developers who need to use IPC should design their applications so that modules
60481 using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the
60482 alternative interfaces.

60483 RATIONALE

60484 None.

60485 FUTURE DIRECTIONS

60486 None.

60487 SEE ALSO

60488 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), *semget()*, *semop()*, *sem_close()*, *sem_destroy()*,
60489 *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_trywait()*, *sem_unlink()*

60490 XBD [Section 4.17](#) (on page 114), [<sys/sem.h>](#)

60491 CHANGE HISTORY

60492 First released in Issue 2. Derived from Issue 2 of the SVID.

60493 Issue 5

60494 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
60495 DIRECTIONS to the APPLICATION USAGE section.

60496 Issue 7

60497 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0532 [345], XSH/TC1-2008/0533 [345],
60498 XSH/TC1-2008/0534 [345], and XSH/TC1-2008/0535 [335] are applied.

60499 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0319 [532] is applied.

60500 **NAME**

60501 semget — get set of XSI semaphores

60502 **SYNOPSIS**

```
60503 XSI #include <sys/sem.h>
60504 int semget(key_t key, int nsems, int semflg);
```

60505 **DESCRIPTION**

60506 The *semget()* function operates on XSI semaphores (see XBD [Section 4.17](#), on page 114). It is
 60507 unspecified whether this function interoperates with the realtime interprocess communication
 60508 facilities defined in [Section 2.8](#) (on page 503).

60509 The *semget()* function shall return the semaphore identifier associated with *key*.

60510 A semaphore identifier with its associated **semid_ds** data structure and its associated set of
 60511 *nsems* semaphores (see **<sys/sem.h>**) is created for *key* if one of the following is true:

60512 The argument *key* is equal to `IPC_PRIVATE`.

60513 The argument *key* does not already have a semaphore identifier associated with it and
 60514 (*semflg* & `IPC_CREAT`) is non-zero.

60515 Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is
 60516 initialized as follows:

60517 In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and
 60518 *sem_perm.gid* shall be set to the effective user ID and effective group ID, respectively, of the
 60519 calling process.

60520 The low-order 9 bits of *sem_perm.mode* shall be set to the low-order 9 bits of *semflg*.

60521 The variable *sem_nsems* shall be set to the value of *nsems*.

60522 The variable *sem_otime* shall be set to 0 and *sem_ctime* shall be set to the current time, as
 60523 described in [Section 2.7.1](#) (on page 502).

60524 The data structure associated with each semaphore in the set need not be initialized. The
 60525 *semctl()* function with the command `SETVAL` or `SETALL` can be used to initialize each
 60526 semaphore.

60527 **RETURN VALUE**

60528 Upon successful completion, *semget()* shall return a non-negative integer, namely a semaphore
 60529 identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

60530 **ERRORS**

60531 The *semget()* function shall fail if:

60532 **[EACCES]** A semaphore identifier exists for *key*, but operation permission as specified by
 60533 the low-order 9 bits of *semflg* would not be granted; see [Section 2.7](#) (on page
 60534 501).

60535 **[EEXIST]** A semaphore identifier exists for the argument *key* but $((semflg \& IPC_CREAT) \& \& (semflg \& IPC_EXCL))$ is non-zero.

60537 **[EINVAL]** The value of *nsems* is either less than or equal to 0 or greater than the system-
 60538 imposed limit, or a semaphore identifier exists for the argument *key*, but the
 60539 number of semaphores in the set associated with it is less than *nsems* and
 60540 *nsems* is not equal to 0.

60541 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*
60542 &IPC_CREAT) is equal to 0.

60543 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the
60544 maximum number of allowed semaphores system-wide would be exceeded.

60545 EXAMPLES

60546 Refer to *semop()*.

60547 APPLICATION USAGE

60548 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
60549 Application developers who need to use IPC should design their applications so that modules
60550 using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the
60551 alternative interfaces.

60552 RATIONALE

60553 None.

60554 FUTURE DIRECTIONS

60555 A future version may require that the value of the *semval*, *sempid*, *semncnt*, and *semzcnt* members
60556 of all semaphores in a semaphore set be initialized to zero when a call to *semget()* creates a
60557 semaphore set. Many semaphore implementations already do this and it greatly simplifies what
60558 an application must do to initialize a semaphore set.

60559 SEE ALSO

60560 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), *ftok()*, *semctl()*, *semop()*, *sem_close()*,
60561 *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_trywait()*, *sem_unlink()*

60562 XBD [Section 4.17](#) (on page 114), [<sys/sem.h>](#)

60563 CHANGE HISTORY

60564 First released in Issue 2. Derived from Issue 2 of the SVID.

60565 Issue 5

60566 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
60567 DIRECTIONS to a new APPLICATION USAGE section.

60568 Issue 6

60569 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/122 is applied, updating the
60570 DESCRIPTION from “each semaphore in the set shall not be initialized” to “each semaphore in
60571 the set need not be initialized”.

60572 Issue 7

60573 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0536 [335,439] and
60574 XSH/TC1-2008/0537 [344] are applied.

60575 **NAME**

60576 semop ‡XSI semaphore operations

60577 **SYNOPSIS**

```
60578 XSI #include <sys/sem.h>
60579 int semop(int semid, struct sembuf *sops, size_t nsops);
```

60580 **DESCRIPTION**

60581 The *semop()* function operates on XSI semaphores (see XBD [Section 4.17](#), on page 114). It is
 60582 unspecified whether this function interoperates with the realtime interprocess communication
 60583 facilities defined in [Section 2.8](#) (on page 503).

60584 The *semop()* function shall perform atomically a user-defined array of semaphore operations in
 60585 array order on the set of semaphores associated with the semaphore identifier specified by the
 60586 argument *semid*.

60587 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The
 60588 implementation shall not modify elements of this array unless the application uses
 60589 implementation-defined extensions.

60590 The argument *nsops* is the number of such structures in the array.

60591 Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
unsigned short	<i>sem_num</i>	Semaphore number.
short	<i>sem_op</i>	Semaphore operation.
short	<i>sem_flg</i>	Operation flags.

60596 Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore
 60597 specified by *semid* and *sem_num*.

60598 The variable *sem_op* specifies one of three semaphore operations:

- 60599 1. If *sem_op* is a negative integer and the calling process has alter permission, one of the
 60600 following shall occur:

60601 If *semval* (see [<sys/sem.h>](#)) is greater than or equal to the absolute value of *sem_op*,
 60602 the absolute value of *sem_op* is subtracted from *semval*. Also, if (*sem_flg*
 60603 &SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the
 60604 *semadj* value of the calling process for the specified semaphore.

60605 If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is
 60606 non-zero, *semop()* shall return immediately.

60607 If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is 0,
 60608 *semop()* shall increment the *semncnt* associated with the specified semaphore and
 60609 suspend execution of the calling thread until one of the following conditions occurs:

60610 ‡If the value of *semval* becomes greater than or equal to the absolute value of
 60611 *sem_op*. When this occurs, the value of *semncnt* associated with the specified
 60612 semaphore shall be decremented, the absolute value of *sem_op* shall be
 60613 subtracted from *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the
 60614 absolute value of *sem_op* shall be added to the *semadj* value of the calling
 60615 process for the specified semaphore.

60616 ‡ If *semid* for which the calling thread is awaiting action is removed from the
 60617 system. When this occurs, *errno* shall be set to [EIDRM] and -1 shall be
 60618 returned.

60619 ‡ If calling thread receives a signal that is to be caught. When this occurs, the
 60620 value of *semcnt* associated with the specified semaphore shall be
 60621 decremented, and the calling thread shall resume execution in the manner
 60622 prescribed in *sigaction()*.

60623 2. If *sem_op* is a positive integer and the calling process has alter permission, the value of
 60624 *sem_op* shall be added to *semval* and, if (*sem_flg* & SEM_UNDO) is non-zero, the value of
 60625 *sem_op* shall be subtracted from the *semadj* value of the calling process for the specified
 60626 semaphore.

60627 3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:

60628 If *semval* is 0, *semop()* shall return immediately.

60629 If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is non-zero, *semop()* shall return
 60630 immediately.

60631 If *semval* is non-zero and (*sem_flg* & IPC_NOWAIT) is 0, *semop()* shall increment the
 60632 *semzcnt* associated with the specified semaphore and suspend execution of the
 60633 calling thread until one of the following occurs:

60634 ‡ If value of *semval* becomes 0, at which time the value of *semzcnt* associated
 60635 with the specified semaphore shall be decremented.

60636 ‡ If *semid* for which the calling thread is awaiting action is removed from the
 60637 system. When this occurs, *errno* shall be set to [EIDRM] and -1 shall be
 60638 returned.

60639 ‡ If calling thread receives a signal that is to be caught. When this occurs, the
 60640 value of *semzcnt* associated with the specified semaphore shall be
 60641 decremented, and the calling thread shall resume execution in the manner
 60642 prescribed in *sigaction()*.

60643 Upon successful completion, the value of *sempid* for each semaphore specified in the array
 60644 pointed to by *sops* shall be set to the process ID of the calling process. Also, the *sem_otime*
 60645 timestamp shall be set to the current time, as described in [Section 2.7.1](#) (on page 502).

60646 RETURN VALUE

60647 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to
 60648 indicate the error.

60649 ERRORS

60650 The *semop()* function shall fail if:

60651 [E2BIG] The value of *nsops* is greater than the system-imposed maximum.

60652 [EACCES] Operation permission is denied to the calling process; see [Section 2.7](#) (on page
 60653 501).

60654 [EAGAIN] The operation would result in suspension of the calling process but (*sem_flg*
 60655 & IPC_NOWAIT) is non-zero.

60656 [EFBIG] The value of *sem_num* is greater than or equal to the number of semaphores in
 60657 the set associated with *semid*.

60658	[EIDRM]	The semaphore identifier <i>semid</i> is removed from the system.
60659	[EINTR]	The <i>semop()</i> function was interrupted by a signal.
60660	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.
60661		
60662		
60663	[ENOSPC]	The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.
60664		
60665	[ERANGE]	An operation would cause a <i>semval</i> to overflow the system-imposed limit, or an operation would cause a <i>semadj</i> value to overflow the system-imposed limit.
60666		
60667		

60668 EXAMPLES

60669 Setting Values in Semaphores

60670 The following example sets the values of the two semaphores associated with the *semid* identifier
60671 to the values contained in the *sb* array.

```
60672 #include <sys/sem.h>
60673 ...
60674 int semid;
60675 struct sembuf sb[2];
60676 int nsops = 2;
60677 int result;

60678 /* Code to initialize semid. */
60679 ...

60680 /* Adjust value of semaphore in the semaphore array semid. */
60681 sb[0].sem_num = 0;
60682 sb[0].sem_op = -1;
60683 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
60684 sb[1].sem_num = 1;
60685 sb[1].sem_op = 1;
60686 sb[1].sem_flg = 0;

60687 result = semop(semid, sb, nsops);
```

60688 Creating a Semaphore Identifier

60689 The following example gets a unique semaphore key using the *ftok()* function, then gets a
60690 semaphore ID associated with that key using the *semget()* function (the first call also tests to
60691 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
60692 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
60693 program attempts to create one semaphore with read/write permission for all. It also uses the
60694 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

60695 After creating the semaphore, the program uses calls to *semctl()* and *semop()* to initialize it to the
60696 values in the *sbuf* array. The number of processes that can execute concurrently without queuing
60697 is initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later
60698 in the program.

60699 Processes that obtain *semid* without creating it check that *sem_otime* is non-zero, to ensure that
60700 the creating process has completed the *semop()* initialization.

```

60701     The final call to semop() acquires the semaphore and waits until it is free; the SEM_UNDO
60702     option releases the semaphore when the process exits, waiting until there are less than two
60703     processes running concurrently.

60704     #include <stdio.h>
60705     #include <sys/sem.h>
60706     #include <sys/stat.h>
60707     #include <errno.h>
60708     #include <stdlib.h>
60709     ...
60710     key_t semkey;
60711     int semid;
60712     struct sembuf sbuf;
60713     union semun {
60714         int val;
60715         struct semid_ds *buf;
60716         unsigned short *array;
60717     } arg;
60718     struct semid_ds ds;
60719     ...
60720     /* Get unique key for semaphore. */
60721     if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
60722         perror("IPC error: ftok"); exit(1);
60723     }

60724     /* Get semaphore ID associated with this key. */
60725     if ((semid = semget(semkey, 0, 0)) == -1) {

60726         /* Semaphore does not exist - Create. */
60727         if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
60728             S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
60729             {
60730                 /* Initialize the semaphore. */
60731                 arg.val = 0;
60732                 sbuf.sem_num = 0;
60733                 sbuf.sem_op = 2; /* This is the number of runs without queuing. */
60734                 sbuf.sem_flg = 0;
60735                 if (semctl(semid, 0, SETVAL, arg) == -1
60736                     || semop(semid, &sbuf, 1) == -1) {
60737                     perror("IPC error: semop"); exit(1);
60738                 }
60739             }
60740         else if (errno == EEXIST) {
60741             if ((semid = semget(semkey, 0, 0)) == -1) {
60742                 perror("IPC error 1: semget"); exit(1);
60743             }
60744             goto check_init;
60745         }
60746         else {
60747             perror("IPC error 2: semget"); exit(1);
60748         }
60749     }
60750     else

```

```

60751     {
60752         /* Check that semid has completed initialization. */
60753         /* An application can use a retry loop at this point rather than
60754            exiting. */
60755         check_init:
60756         arg.buf = &ds;
60757         if (semctl(semid, 0, IPC_STAT, arg) < 0) {
60758             perror("IPC error 3: semctl"); exit(1);
60759         }
60760         if (ds.sem_otime == 0) {
60761             perror("IPC error 4: semctl"); exit(1);
60762         }
60763     }
60764     ...
60765     sbuf.sem_num = 0;
60766     sbuf.sem_op = -1;
60767     sbuf.sem_flg = SEM_UNDO;
60768     if (semop(semid, &sbuf, 1) == -1) {
60769         perror("IPC Error: semop"); exit(1);
60770     }

```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*, *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_trywait()*, *sem_unlink()*

XBD [Section 4.17](#) (on page 114), [<sys/ipc.h>](#), [<sys/sem.h>](#), [<sys/types.h>](#)

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 7

SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the operations in *sops* will be performed when there are multiple operations.

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0538 [329,429], XSH/TC1-2008/0539 [345,428], XSH/TC1-2008/0540 [329,429], XSH/TC1-2008/0541 [335], and XSH/TC1-2008/0542 [291,429] are applied.

60796 **NAME**

60797 send †'send a message on a socket

60798 **SYNOPSIS**

60799 #include <sys/socket.h>

60800 ssize_t send(int *socket*, const void **buffer*, size_t *length*, int *flags*);60801 **DESCRIPTION**

60802 The *send()* function shall initiate transmission of a message from the specified socket to its peer.
 60803 The *send()* function shall send a message only when the socket is connected. If the socket is a
 60804 connectionless-mode socket, the message shall be sent to the pre-specified peer address.

60805 The *send()* function takes the following arguments:60806 *socket* Specifies the socket file descriptor.60807 *buffer* Points to the buffer containing the message to send.60808 *length* Specifies the length of the message in bytes.60809 *flags* Specifies the type of message transmission. Values of this argument are
 60810 formed by logically OR'ing zero or more of the following flags:

60811 MSG_EOR Terminates a record (if supported by the protocol).

60812 MSG_OOB Sends out-of-band data on sockets that support out-of-
 60813 band communications. The significance and semantics
 60814 of out-of-band data are protocol-specific.60815 MSG_NOSIGNAL Requests not to send the SIGPIPE signal if an attempt to
 60816 send is made on a stream-oriented socket that is no
 60817 longer connected. The [EPIPE] error shall still be
 60818 returned.60819 The length of the message to be sent is specified by the *length* argument. If the message is too
 60820 long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.60821 Successful completion of a call to *send()* does not guarantee delivery of the message. A return
 60822 value of -1 indicates only locally-detected errors.60823 If space is not available at the sending socket to hold the message to be transmitted, and the
 60824 socket file descriptor does not have O_NONBLOCK set, *send()* shall block until space is
 60825 available. If space is not available at the sending socket to hold the message to be transmitted,
 60826 and the socket file descriptor does have O_NONBLOCK set, *send()* shall fail. The *select()* and
 60827 *poll()* functions can be used to determine when it is possible to send more data.60828 The socket in use may require the process to have appropriate privileges to use the *send()*
 60829 function.60830 **RETURN VALUE**60831 Upon successful completion, *send()* shall return the number of bytes sent. Otherwise, -1 shall be
 60832 returned and *errno* set to indicate the error.60833 **ERRORS**60834 The *send()* function shall fail if:

60835 [EAGAIN] or [EWOULDBLOCK]

60836 The socket's file descriptor is marked O_NONBLOCK and the requested
 60837 operation would block.

60838	[EBADF]	The <i>socket</i> argument is not a valid file descriptor.
60839	[ECONNRESET]	A connection was forcibly closed by a peer.
60840	[EDESTADDRREQ]	
60841		The socket is not connection-mode and no peer address is set.
60842	[EINTR]	A signal interrupted <i>send()</i> before any data was transmitted.
60843	[EMSGSIZE]	The message is too large to be sent all at once, as the socket requires.
60844	[ENOTCONN]	The socket is not connected.
60845	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
60846	[EOPNOTSUPP]	The <i>socket</i> argument is associated with a socket that does not support one or more of the values set in <i>flags</i> .
60847		
60848	[EPIPE]	The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.
60849		
60850		
60851		
60852		The <i>send()</i> function may fail if:
60853	[EACCES]	The calling process does not have appropriate privileges.
60854	[EIO]	An I/O error occurred while reading from or writing to the file system.
60855	[ENETDOWN]	The local network interface used to reach the destination is down.
60856	[ENETUNREACH]	
60857		No route to the network is present.
60858	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.

EXAMPLES

60859 None.

APPLICATION USAGE

60862 If the *socket* argument refers to a connection-mode socket, the *send()* function is equivalent to *sendto()* (with any value for the *dest_addr* and *dest_len* arguments, as they are ignored in this case). If the *socket* argument refers to a socket and the *flags* argument is 0, the *send()* function is equivalent to *write()*.

RATIONALE

60866 None.

FUTURE DIRECTIONS

60868 None.

SEE ALSO

60871 [connect\(\)](#), [getsockopt\(\)](#), [poll\(\)](#), [pselect\(\)](#), [recv\(\)](#), [recvfrom\(\)](#), [recvmsg\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#),
60872 [setsockopt\(\)](#), [shutdown\(\)](#), [socket\(\)](#), [write\(\)](#)

60873 XBD [<sys/socket.h>](#)

CHANGE HISTORY

60874 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

60875

60876 **Issue 7**

60877 Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify
60878 the behavior when the socket is a connectionless-mode socket.

60879 The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended
60880 API Set Part 2.

60881 The [EPIPE] error is modified.

60882 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0543 [463] is applied.

60883 **NAME**

60884 sendmsg — send a message on a socket using a message structure

60885 **SYNOPSIS**

60886 #include <sys/socket.h>

60887 ssize_t sendmsg(int socket, const struct msghdr *message, int flags);

60888 **DESCRIPTION**

60889 The *sendmsg()* function shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified in **msghdr** (overriding the pre-specified peer address), or the function shall return `-1` and set *errno* to [EISCONN]. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

60896 The *sendmsg()* function takes the following arguments:

60897	<i>socket</i>	Specifies the socket file descriptor.
60898	<i>message</i>	Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.
60901	<i>flags</i>	Specifies the type of message transmission. The application may specify 0 or the following flag:
60903	MSG_EOR	Terminates a record (if supported by the protocol).
60904	MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
60907	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.

60911 The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg_iov* is sent in turn.

60916 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A return value of `-1` indicates only locally-detected errors.

60918 If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have `O_NONBLOCK` set, the *sendmsg()* function shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have `O_NONBLOCK` set, the *sendmsg()* function shall fail.

60923 If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg()* shall fail if the `SO_BROADCAST` option is not set for the socket.

60925 The socket in use may require the process to have appropriate privileges to use the *sendmsg()* function.

60927 **RETURN VALUE**

60928 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise, `-1`
 60929 shall be returned and *errno* set to indicate the error.

60930 **ERRORS**

60931 The *sendmsg()* function shall fail if:

60932 [EAGAIN] or [EWOULDBLOCK]

60933 The socket's file descriptor is marked `O_NONBLOCK` and the requested
 60934 operation would block.

60935 [EAFNOSUPPORT]

60936 Addresses in the specified address family cannot be used with this socket.

60937 [EBADF] The *socket* argument is not a valid file descriptor.

60938 [ECONNRESET] A connection was forcibly closed by a peer.

60939 [EINTR] A signal interrupted *sendmsg()* before any data was transmitted.

60940 [EINVAL] The sum of the *iov_len* values overflows an `ssize_t`.

60941 [EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the
 60942 *msg_iovlen* member of the `msghdr` structure pointed to by *message* is less than
 60943 or equal to 0 or is greater than `{IOV_MAX}`.

60944 [ENOTCONN] The socket is connection-mode but is not connected.

60945 [ENOTSOCK] The *socket* argument does not refer to a socket.

60946 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
 60947 more of the values set in *flags*.

60948 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
 60949 no longer connected. In the latter case, and if the socket is of type
 60950 `SOCK_STREAM` or `SOCK_SEQPACKET` and the `MSG_NOSIGNAL` flag is not
 60951 set, the `SIGPIPE` signal is generated to the calling thread.

60952 If the address family of the socket is `AF_UNIX`, then *sendmsg()* shall fail if:

60953 [EIO] An I/O error occurred while reading from or writing to the file system.

60954 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
 60955 in the socket address.

60956 [ENAMETOOLONG]

60957 The length of a component of a pathname is longer than `{NAME_MAX}`.

60958 [ENOENT] A component of the pathname does not name an existing file or the path name
 60959 is an empty string.

60960 [ENOTDIR] A component of the path prefix of the pathname in the socket address names
 60961 an existing file that is neither a directory nor a symbolic link to a directory, or
 60962 the pathname in the socket address contains at least one non-`<slash>` character
 60963 and ends with one or more trailing `<slash>` characters and the last pathname
 60964 component names an existing file that is neither a directory nor a symbolic
 60965 link to a directory.

60966 The *sendmsg()* function may fail if:

60967 [EACCES] Search permission is denied for a component of the path prefix; or write access
60968 to the named socket is denied.

60969 [EDESTADDRREQ]
60970 The socket is not connection-mode and does not have its peer address set, and
60971 no destination address was specified.

60972 [EHOSTUNREACH]
60973 The destination host cannot be reached (probably because the host is down or
60974 a remote router cannot reach it).

60975 [EIO] An I/O error occurred while reading from or writing to the file system.

60976 [EISCONN] A destination address was specified and the socket is already connected.

60977 [ENETDOWN] The local network interface used to reach the destination is down.

60978 [ENETUNREACH]
60979 No route to the network is present.

60980 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

60981 [ENOMEM] Insufficient memory was available to fulfill the request.

60982 If the address family of the socket is AF_UNIX, then *sendmsg()* may fail if:

60983 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
60984 resolution of the pathname in the socket address.

60985 [ENAMETOOLONG]
60986 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
60987 symbolic link produced an intermediate result with a length that exceeds
60988 {PATH_MAX}.

60989 EXAMPLES

60990 Done.

60991 APPLICATION USAGE

60992 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

60993 RATIONALE

60994 None.

60995 FUTURE DIRECTIONS

60996 None.

60997 SEE ALSO

60998 *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*,
60999 *shutdown()*, *socket()*

61000 XBD <[sys/socket.h](#)>

61001 CHANGE HISTORY

61002 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

61003 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
61004 [ELOOP] error condition is added.

61005 **Issue 7**

- 61006 Austin Group Interpretation 1003.1-2001 #073 is applied, updating the DESCRIPTION.
- 61007 Austin Group Interpretation 1003.1-2001 #143 is applied.
- 61008 The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
- 61009
- 61010 The [EPIPE] error is modified.
- 61011 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0544 [324] is applied.

61012 **NAME**

61013 sendto ✚send a message on a socket

61014 **SYNOPSIS**

```
61015        #include <sys/socket.h>
61016        ssize_t sendto(int socket, const void *message, size_t length,
61017                       int flags, const struct sockaddr *dest_addr,
61018                       socklen_t dest_len);
```

61019 **DESCRIPTION**

61020 The *sendto()* function shall send a message through a connection-mode or connectionless-mode
61021 socket.

61022 If the socket is a connectionless-mode socket, the message shall be sent to the address specified
61023 by *dest_addr* if no pre-specified peer address has been set. If a peer address has been pre-
61024 specified, either the message shall be sent to the address specified by *dest_addr* (overriding the
61025 pre-specified peer address), or the function shall return -1 and set *errno* to [EISCONN].

61026 If the socket is connection-mode, *dest_addr* shall be ignored.

61027 The *sendto()* function takes the following arguments:

61028	<i>socket</i>	Specifies the socket file descriptor.
61029	<i>message</i>	Points to a buffer containing the message to be sent.
61030	<i>length</i>	Specifies the size of the message in bytes.
61031	<i>flags</i>	Specifies the type of message transmission. Values of this argument are 61032 formed by logically OR'ing zero or more of the following flags:
61033	MSG_EOR	Terminates a record (if supported by the protocol).
61034	MSG_OOB	Sends out-of-band data on sockets that support out-of- 61035 band data. The significance and semantics of out-of- 61036 band data are protocol-specific.
61037	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to 61038 send is made on a stream-oriented socket that is no 61039 longer connected. The [EPIPE] error shall still be 61040 returned.
61041	<i>dest_addr</i>	Points to a sockaddr structure containing the destination address. The length 61042 and format of the address depend on the address family of the socket.
61043	<i>dest_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>dest_addr</i> 61044 argument.

61045 If the socket protocol supports broadcast and the specified address is a broadcast address for the
61046 socket protocol, *sendto()* shall fail if the SO_BROADCAST option is not set for the socket.

61047 The *dest_addr* argument specifies the address of the target.

61048 The *length* argument specifies the length of the message.

61049 Successful completion of a call to *sendto()* does not guarantee delivery of the message. A return
61050 value of -1 indicates only locally-detected errors.

61051 If space is not available at the sending socket to hold the message to be transmitted and the
61052 socket file descriptor does not have O_NONBLOCK set, *sendto()* shall block until space is
61053 available. If space is not available at the sending socket to hold the message to be transmitted

- 61054 and the socket file descriptor does have O_NONBLOCK set, *sendto()* shall fail.
- 61055 The socket in use may require the process to have appropriate privileges to use the *sendto()*
61056 function.
- 61057 **RETURN VALUE**
- 61058 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise, -1 shall
61059 be returned and *errno* set to indicate the error.
- 61060 **ERRORS**
- 61061 The *sendto()* function shall fail if:
- 61062 [EAFNOSUPPORT]
61063 Addresses in the specified address family cannot be used with this socket.
- 61064 [EAGAIN] or [EWOULDBLOCK]
61065 The socket's file descriptor is marked O_NONBLOCK and the requested
61066 operation would block.
- 61067 [EBADF] The *socket* argument is not a valid file descriptor.
- 61068 [ECONNRESET] A connection was forcibly closed by a peer.
- 61069 [EINTR] A signal interrupted *sendto()* before any data was transmitted.
- 61070 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 61071 [ENOTCONN] The socket is connection-mode but is not connected.
- 61072 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 61073 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
61074 more of the values set in *flags*.
- 61075 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
61076 no longer connected. In the latter case, and if the socket is of type
61077 SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not
61078 set, the SIGPIPE signal is generated to the calling thread.
- 61079 If the address family of the socket is AF_UNIX, then *sendto()* shall fail if:
- 61080 [EIO] An I/O error occurred while reading from or writing to the file system.
- 61081 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
61082 in the socket address.
- 61083 [ENAMETOOLONG]
61084 The length of a component of a pathname is longer than {NAME_MAX}.
- 61085 [ENOENT] A component of the pathname does not name an existing file or the pathname
61086 is an empty string.
- 61087 [ENOTDIR] A component of the path prefix of the pathname in the socket address names
61088 an existing file that is neither a directory nor a symbolic link to a directory, or
61089 the pathname in the socket address contains at least one non-*<slash>* character
61090 and ends with one or more trailing *<slash>* characters and the last pathname
61091 component names an existing file that is neither a directory nor a symbolic
61092 link to a directory.

- 61093 The *sendto()* function may fail if:
- 61094 [EACCES] Search permission is denied for a component of the path prefix; or write access
61095 to the named socket is denied.
- 61096 [EDESTADDRREQ]
61097 The socket is not connection-mode and does not have its peer address set, and
61098 no destination address was specified.
- 61099 [EHOSTUNREACH]
61100 The destination host cannot be reached (probably because the host is down or
61101 a remote router cannot reach it).
- 61102 [EINVAL] The *dest_len* argument is not a valid length for the address family.
- 61103 [EIO] An I/O error occurred while reading from or writing to the file system.
- 61104 [EISCONN] A destination address was specified and the socket is already connected.
- 61105 [ENETDOWN] The local network interface used to reach the destination is down.
- 61106 [ENETUNREACH]
61107 No route to the network is present.
- 61108 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 61109 [ENOMEM] Insufficient memory was available to fulfill the request.
- 61110 If the address family of the socket is AF_UNIX, then *sendto()* may fail if:
- 61111 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
61112 resolution of the pathname in the socket address.
- 61113 [ENAMETOOLONG]
61114 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
61115 symbolic link produced an intermediate result with a length that exceeds
61116 {PATH_MAX}.
- 61117 **EXAMPLES**
61118 None.
- 61119 **APPLICATION USAGE**
61120 The *select()* and *poll()* functions can be used to determine when it is possible to send more data.
- 61121 **RATIONALE**
61122 None.
- 61123 **FUTURE DIRECTIONS**
61124 None.
- 61125 **SEE ALSO**
61126 *getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *setsockopt()*,
61127 *shutdown()*, *socket()*
- 61128 XBD <[sys/socket.h](#)>
- 61129 **CHANGE HISTORY**
61130 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.
- 61131 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
61132 [ELOOP] error condition is added.

61133 **Issue 7**

61134 Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, updating the [EISCONN]
61135 error and the DESCRIPTION.

61136 Austin Group Interpretation 1003.1-2001 #143 is applied, clarifying the [ENAMETOOLONG]
61137 error condition.

61138 The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended
61139 API Set Part 2.

61140 The [EPIPE] error is modified.

61141 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0545 [324] is applied.

61142 **NAME**

61143 setbuf — assign buffering to a stream

61144 **SYNOPSIS**

61145 #include <stdio.h>

61146 void setbuf(FILE *restrict stream, char *restrict buf);

61147 **DESCRIPTION**

61148 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 61149 conflict between the requirements described here and the ISO C standard is unintentional. This
 61150 volume of POSIX.1-2017 defers to the ISO C standard.

61151 Except that it returns no value, the function call:

61152 setbuf(stream, buf)

61153 shall be equivalent to:

61154 setvbuf(stream, buf, _IOFBF, BUFSIZ)

61155 if *buf* is not a null pointer, or to:

61156 setvbuf(stream, buf, _IONBF, BUFSIZ)

61157 if *buf* is a null pointer.61158 **RETURN VALUE**61159 The *setbuf()* function shall not return a value.61160 **ERRORS**

61161 Although the *setvbuf()* interface may set *errno* in defined ways, the value of *errno* after a call to
 61162 *setbuf()* is unspecified.

61163 **EXAMPLES**

61164 None.

61165 **APPLICATION USAGE**

61166 A common source of error is allocating buffer space as an “automatic” variable in a code block,
 61167 and then failing to close the stream in the same block.

61168 With *setbuf()*, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ
 61169 bytes are used for the buffer area.

61170 Since *errno* is not required to be unchanged on success, in order to correctly detect and possibly
 61171 recover from errors, applications should use *setvbuf()* instead of *setbuf()*.

61172 **RATIONALE**

61173 None.

61174 **FUTURE DIRECTIONS**

61175 None.

61176 **SEE ALSO**61177 [Section 2.5](#) (on page 495), *fopen()*, *setvbuf()*61178 XBD [<stdio.h>](#)61179 **CHANGE HISTORY**

61180 First released in Issue 1. Derived from Issue 1 of the SVID.

61181 **Issue 6**

61182 The prototype for *setbuf()* is updated for alignment with the ISO/IEC 9899:1999 standard.

61183 **Issue 7**

61184 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0546 [397], XSH/TC1-2008/0547 [397],
61185 and XSH/TC1-2008/0548 [14] are applied.

61186 **NAME**

61187 setegid — set the effective group ID

61188 **SYNOPSIS**

61189 #include <unistd.h>

61190 int setegid(gid_t gid);

61191 **DESCRIPTION**

61192 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate
61193 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group
61194 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

61195 The *setegid()* function shall not affect the supplementary group list in any way.

61196 **RETURN VALUE**

61197 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
61198 indicate the error.

61199 **ERRORS**

61200 The *setegid()* function shall fail if:

61201 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
61202 implementation.

61203 [EPERM] The process does not have appropriate privileges and *gid* does not match the
61204 real group ID or the saved set-group-ID.

61205 **EXAMPLES**

61206 None.

61207 **APPLICATION USAGE**

61208 None.

61209 **RATIONALE**

61210 Refer to the RATIONALE section in *setuid()*.

61211 **FUTURE DIRECTIONS**

61212 None.

61213 **SEE ALSO**

61214 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

61215 XBD <sys/types.h>, <unistd.h>

61216 **CHANGE HISTORY**

61217 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

61218 **NAME**

61219 setenv — add or change environment variable

61220 **SYNOPSIS**

```
61221 CX #include <stdlib.h>
61222 int setenv(const char *envname, const char *envval, int overwrite);
```

61223 **DESCRIPTION**

61224 The *setenv()* function shall update or add a variable in the environment of the calling process.
 61225 The *envname* argument points to a string containing the name of an environment variable to be
 61226 added or altered. The environment variable shall be set to the value to which *envval* points. The
 61227 function shall fail if *envname* points to a string which contains an '=' character. If the
 61228 environment variable named by *envname* already exists and the value of *overwrite* is non-zero,
 61229 the function shall return success and the environment shall be updated. If the environment
 61230 variable named by *envname* already exists and the value of *overwrite* is zero, the function shall
 61231 return success and the environment shall remain unchanged.

61232 The *setenv()* function shall update the list of pointers to which *environ* points.

61233 The strings described by *envname* and *envval* are copied by this function.

61234 The *setenv()* function need not be thread-safe.

61235 **RETURN VALUE**

61236 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to
 61237 indicate the error, and the environment shall be unchanged.

61238 **ERRORS**

61239 The *setenv()* function shall fail if:

61240 [EINVAL] The *envname* argument points to an empty string or points to a string
 61241 containing an '=' character.

61242 [ENOMEM] Insufficient memory was available to add a variable or its value to the
 61243 environment.

61244 **EXAMPLES**

61245 None.

61246 **APPLICATION USAGE**

61247 See *exec()* for restrictions on changing the environment in multi-threaded applications.

61248 **RATIONALE**

61249 Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular, if
 61250 the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an
 61251 obsolete copy of the environment (as may any other copy of *environ*). However, other than the
 61252 aforementioned restriction, the standard developers intended that the traditional method of
 61253 walking through the environment by way of the *environ* pointer must be supported.

61254 It was decided that *setenv()* should be required by this version because it addresses a piece of
 61255 missing functionality, and does not impose a significant burden on the implementor.

61256 There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*
 61257 function should be required as a mandatory function. The *setenv()* function was chosen because
 61258 it permitted the implementation of the *unsetenv()* function to delete environmental variables,
 61259 without specifying an additional interface. The *putenv()* function is available as part of the XSI
 61260 option.

61261 The standard developers considered requiring that *setenv()* indicate an error when a call to it
61262 would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might
61263 be temporary, with the application eventually reducing the environment size. The ultimate
61264 success or failure depends on the size at the time of a call to *exec*, which returns an indication of
61265 this error condition.

61266 See also the RATIONALE section in *getenv()*.

61267 FUTURE DIRECTIONS

61268 None.

61269 SEE ALSO

61270 *exec*, *getenv()*, *putenv()*, *unsetenv()*

61271 XBD [<stdlib.h>](#), [<sys/types.h>](#), [<unistd.h>](#)

61272 CHANGE HISTORY

61273 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

61274 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in
61275 the APPLICATION USAGE and SEE ALSO sections.

61276 Issue 7

61277 Austin Group Interpretation 1003.1-2001 #156 is applied.

61278 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0549 [167], XSH/TC1-2008/0550 [185],
61279 XSH/TC1-2008/0551 [167], and XSH/TC1-2008/0552 [38] are applied.

61280 NAME

61281 seteuid — set effective user ID

61282 SYNOPSIS

61283 #include <unistd.h>
61284 int seteuid(uid_t uid);

61285 DESCRIPTION

61286 If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate
61287 privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID
61288 and saved set-user-ID shall remain unchanged.

61289 The *seteuid()* function shall not affect the supplementary group list in any way.

61290 RETURN VALUE

61291 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
61292 indicate the error.

61293 ERRORS

61294 The *seteuid()* function shall fail if:

61295 [EINVAL] The value of the *uid* argument is invalid and is not supported by the
61296 implementation.

61297 [EPERM] The process does not have appropriate privileges and *uid* does not match the
61298 real user ID or the saved set-user-ID.

61299 EXAMPLES

61300 None.

61301 APPLICATION USAGE

61302 None.

61303 RATIONALE

61304 Refer to the RATIONALE section in *setuid()*.

61305 FUTURE DIRECTIONS

61306 None.

61307 SEE ALSO

61308 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

61309 XBD <sys/types.h>, <unistd.h>

61310 CHANGE HISTORY

61311 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

61312 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/123 is applied, making an editorial
61313 correction to the [EPERM] error in the ERRORS section.

61314 **NAME**

61315 setgid — set-group-ID

61316 **SYNOPSIS**

61317 #include <unistd.h>

61318 int setgid(gid_t gid);

61319 **DESCRIPTION**61320 If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID,
61321 and the saved set-group-ID of the calling process to *gid*.61322 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the
61323 saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved
61324 set-group-ID shall remain unchanged.61325 The *setgid()* function shall not affect the supplementary group list in any way.

61326 Any supplementary group IDs of the calling process shall remain unchanged.

61327 **RETURN VALUE**61328 Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to
61329 indicate the error.61330 **ERRORS**61331 The *setgid()* function shall fail if:61332 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
61333 implementation.61334 [EPERM] The process does not have appropriate privileges and *gid* does not match the
61335 real group ID or the saved set-group-ID.61336 **EXAMPLES**

61337 None.

61338 **APPLICATION USAGE**

61339 None.

61340 **RATIONALE**61341 Refer to the RATIONALE section in *setuid()*.61342 **FUTURE DIRECTIONS**

61343 None.

61344 **SEE ALSO**61345 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*

61346 XBD <sys/types.h>, <unistd.h>

61347 **CHANGE HISTORY**

61348 First released in Issue 1. Derived from Issue 1 of the SVID.

61349 **Issue 6**

61350 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

61351 The following new requirements on POSIX implementations derive from alignment with the
61352 Single UNIX Specification:61353 The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
61354 required for conforming implementations of previous POSIX specifications, it was not
61355 required for UNIX applications.

61356 Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS
61357 requirement.

61358 The following changes were made to align with the IEEE P1003.1a draft standard:

61359 The effects of `setgid()` in processes without appropriate privileges are changed.

61360 A requirement that the supplementary group list is not affected is added.

61361 **NAME**

61362 setgrent — reset the group database to the first entry

61363 **SYNOPSIS**

61364 XSI #include <grp.h>

61365 void setgrent(void);

61366 **DESCRIPTION**

61367 Refer to *endgrent()*.

61368 **NAME**
61369 sethostent †network host database functions

61370 **SYNOPSIS**
61371 #include <netdb.h>
61372 void sethostent(int stayopen);

61373 **DESCRIPTION**
61374 Refer to *endhostent()*.

61375 **NAME**

61376 setitimer ‡set the value of an interval timer

61377 **SYNOPSIS**

```
61378 OB XSI #include <sys/time.h>
61379       int setitimer(int which, const struct itimerval *restrict value,
61380                    struct itimerval *restrict ovalue);
```

61381 **DESCRIPTION**

61382 Refer to [getitimer\(\)](#).

61383 **NAME**

61384 setjmp ‡set jump point for a non-local goto

61385 **SYNOPSIS**

61386 #include <setjmp.h>

61387 int setjmp(jmp_buf env);

61388 **DESCRIPTION**

61389 CX The functionality described on this reference page is aligned with the ISO C standard. Any
61390 conflict between the requirements described here and the ISO C standard is unintentional. This
61391 volume of POSIX.1-2017 defers to the ISO C standard.

61392 A call to *setjmp()* shall save the calling environment in its *env* argument for later use by
61393 *longjmp()*.

61394 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in
61395 order to access an actual function, or a program defines an external identifier with the name
61396 *setjmp*, the behavior is undefined.

61397 An application shall ensure that an invocation of *setjmp()* appears in one of the following
61398 contexts only:

61399 The entire controlling expression of a selection or iteration statement

61400 One operand of a relational or equality operator with the other operand an integral
61401 constant expression, with the resulting expression being the entire controlling expression
61402 of a selection or iteration statement

61403 The operand of a unary '!' operator with the resulting expression being the entire
61404 controlling expression of a selection or iteration

61405 The entire expression of an expression statement (possibly cast to **void**)

61406 If the invocation appears in any other context, the behavior is undefined.

61407 **RETURN VALUE**

61408 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to
61409 *longjmp()*, *setjmp()* shall return a non-zero value.

61410 **ERRORS**

61411 No errors are defined.

61412 **EXAMPLES**

61413 None.

61414 **APPLICATION USAGE**

61415 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-
61416 level subroutine of a program.

61417 **RATIONALE**

61418 None.

61419 **FUTURE DIRECTIONS**

61420 None.

61421 **SEE ALSO**

61422 [longjmp\(\)](#), [sigsetjmp\(\)](#)

61423 XBD [<setjmp.h>](#)

61424 **CHANGE HISTORY**

61425 First released in Issue 1. Derived from Issue 1 of the SVID.

61426 **Issue 6**

61427 The normative text is updated to avoid use of the term “must” for application requirements.

61428 **NAME**

61429 setkey ‡set encoding key(CRYPT)

61430 **SYNOPSIS**

```
61431 XSI #include <stdlib.h>
61432 void setkey(const char *key);
```

61433 **DESCRIPTION**

61434 The *setkey()* function provides access to an implementation-defined encoding algorithm. The
 61435 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical
 61436 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is
 61437 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used
 61438 with the algorithm to encode a string *block* passed to *encrypt()*.

61439 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to
 61440 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on
 61441 return, an error has occurred.

61442 The *setkey()* function need not be thread-safe.

61443 **RETURN VALUE**

61444 No values are returned.

61445 **ERRORS**61446 The *setkey()* function shall fail if:

61447 [ENOSYS] The functionality is not supported on this implementation.

61448 **EXAMPLES**

61449 None.

61450 **APPLICATION USAGE**

61451 Decoding need not be implemented in all environments. This is related to government
 61452 restrictions in some countries on encryption and decryption routines. Historical practice has
 61453 been to ship a different version of the encryption library without the decryption feature in the
 61454 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

61455 **RATIONALE**

61456 None.

61457 **FUTURE DIRECTIONS**

61458 A future version of the standard may mark this interface as obsolete or remove it altogether.

61459 **SEE ALSO**61460 *crypt()*, *encrypt()*

61461 XBD <stdlib.h>

61462 **CHANGE HISTORY**

61463 First released in Issue 1. Derived from Issue 1 of the SVID.

61464 **Issue 5**61465 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.61466 **Issue 7**

61467 Austin Group Interpretation 1003.1-2001 #156 is applied.

61468 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0320 [899] is applied.

61469 **NAME**

61470 setlocale — set program locale

61471 **SYNOPSIS**

61472 #include <locale.h>

61473 char *setlocale(int category, const char *locale);

61474 **DESCRIPTION**

61475 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 61476 conflict between the requirements described here and the ISO C standard is unintentional. This
 61477 volume of POSIX.1-2017 defers to the ISO C standard.

61478 The *setlocale()* function selects the appropriate piece of the global locale, as specified by the
 61479 *category* and *locale* arguments, and can be used to change or query the entire global locale or
 61480 portions thereof. The value LC_ALL for *category* names the entire global locale; other values for
 61481 *category* name only a part of the global locale:

61482 LC_COLLATE Affects the behavior of regular expressions and the collation functions.

61483 LC_CTYPE Affects the behavior of regular expressions, character classification, character
 61484 conversion functions, and wide-character functions.

61485 CX LC_MESSAGES Affects the affirmative and negative response expressions returned by
 61486 *nl_langinfo()* and the way message catalogs are located. It may also affect the
 61487 behavior of functions that return or write message strings.

61488 LC_MONETARY Affects the behavior of functions that handle monetary values.

61489 LC_NUMERIC Affects the behavior of functions that handle numeric values.

61490 LC_TIME Affects the behavior of the time conversion functions.

61491 The *locale* argument is a pointer to a character string containing the required setting of *category*.
 61492 The contents of this string are implementation-defined. In addition, the following preset values
 61493 of *locale* are defined for all settings of *category*:

61494 CX "POSIX" Specifies the minimal environment for C-language translation called the
 61495 POSIX locale. The POSIX locale is the default global locale at entry to *main()*.

61496 "C" Equivalent to "POSIX".

61497 CX "" Specifies an implementation-defined native environment. The determination
 61498 of the name of the new locale for the specified category depends on the value
 61499 of the associated environment variables, *LC_** and *LANG*; see XBD Chapter 7
 61500 (on page 135) and Chapter 8 (on page 173).

61501 A null pointer Directs *setlocale()* to query the current global locale setting and return the
 61502 name of the locale if *category* is not LC_ALL, or a string which encodes the
 61503 locale name(s) for all of the individual categories if *category* is LC_ALL.

61504 CX Setting all of the categories of the global locale is similar to successively setting each individual
 61505 category of the global locale, except that all error checking is done before any actions are
 61506 performed. To set all the categories of the global locale, *setlocale()* can be invoked as:

```
61507 setlocale(LC_ALL, "");
```

61508 In this case, *setlocale()* shall first verify that the values of all the environment variables it needs
 61509 according to the precedence rules (described in XBD Chapter 8, on page 173) indicate supported
 61510 locales. If the value of any of these environment variable searches yields a locale that is not
 61511 supported (and non-null), *setlocale()* shall return a null pointer and the global locale shall not be

61512 changed. If all environment variables name supported locales, *setlocale()* shall proceed as if it
 61513 had been called for each category, using the appropriate value from the associated environment
 61514 variable or from the implementation-defined default if there is no such value.

61515 The global locale established using *setlocale()* shall only be used in threads for which no current
 61516 locale has been set using *uselocale()* or whose current locale has been set to the global locale
 61517 using *uselocale(LC_GLOBAL_LOCALE)*.

61518 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 61519 *setlocale()*.

61520 CX The *setlocale()* function need not be thread-safe.

61521 RETURN VALUE

61522 Upon successful completion, *setlocale()* shall return the string associated with the specified
 61523 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the global locale
 61524 shall not be changed.

61525 A null pointer for *locale* shall cause *setlocale()* to return a pointer to the string associated with the
 61526 specified *category* for the current global locale. The global locale shall not be changed.

61527 The string returned by *setlocale()* is such that a subsequent call with that string and its associated
 61528 *category* shall restore that part of the global locale. The application shall not modify the string
 61529 CX returned. The returned string pointer might be invalidated or the string content might be
 61530 CX overwritten by a subsequent call to *setlocale()*. The returned pointer might also be invalidated if
 61531 the calling thread is terminated.

61532 ERRORS

61533 No errors are defined.

61534 EXAMPLES

61535 None.

61536 APPLICATION USAGE

61537 The following code illustrates how a program can initialize the international environment for
 61538 one language, while selectively modifying the global locale such that regular expressions and
 61539 string operations can be applied to text recorded in a different language:

```
61540 setlocale(LC_ALL, "De");
61541 setlocale(LC_COLLATE, "Fr@dict");
```

61542 Internationalized programs can initiate language operation according to environment variable
 61543 settings (see XBD Section 8.2, on page 174) by calling *setlocale()* as follows:

```
61544 setlocale(LC_ALL, "");
```

61545 Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened
 61546 by calls to *catopen()*.

61547 In order to make use of different locale settings while multiple threads are running, applications
 61548 should use *uselocale()* in preference to *setlocale()*.

61549 RATIONALE

61550 References to the international environment or locale in the following text relate to the global
 61551 locale for the process. This can be overridden for individual threads using *uselocale()*.

61552 The ISO C standard defines a collection of functions to support internationalization. One of the
 61553 most significant aspects of these functions is a facility to set and query the *international*
 61554 *environment*. The international environment is a repository of information that affects the
 61555 behavior of certain functionality, namely:

- 61556 1. Character handling
- 61557 2. Collating
- 61558 3. Date/time formatting
- 61559 4. Numeric editing
- 61560 5. Monetary formatting
- 61561 6. Messaging

61562 The `setlocale()` function provides the application developer with the ability to set all or portions,
 61563 called *categories*, of the international environment. These categories correspond to the areas of
 61564 functionality mentioned above. The syntax for `setlocale()` is as follows:

```
61565 char *setlocale(int category, const char *locale);
```

61566 where *category* is the name of one of following categories, namely:

```
61567     LC_COLLATE
61568     LC_CTYPE
61569     LC_MESSAGES
61570     LC_MONETARY
61571     LC_NUMERIC
61572     LC_TIME
```

61573 In addition, a special value called `LC_ALL` directs `setlocale()` to set all categories.

61574 There are two primary uses of `setlocale()`:

- 61575 1. Querying the international environment to find out what it is set to
- 61576 2. Setting the international environment, or *locale*, to a specific value

61577 The behavior of `setlocale()` in these two areas is described below. Since it is difficult to describe
 61578 the behavior in words, examples are used to illustrate the behavior of specific uses.

61579 To query the international environment, `setlocale()` is invoked with a specific category and the
 61580 null pointer as the locale. The null pointer is a special directive to `setlocale()` that tells it to query
 61581 rather than set the international environment. The following syntax is used to query the name of
 61582 the international environment:

```
61583 setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
61584          LC_NUMERIC, LC_TIME}, (char *) NULL);
```

61585 The `setlocale()` function shall return the string corresponding to the current international
 61586 environment. This value may be used by a subsequent call to `setlocale()` to reset the international
 61587 environment to this value. However, it should be noted that the return value from `setlocale()`
 61588 may be a pointer to a static area within the function and is not guaranteed to remain unchanged
 61589 (that is, it may be modified by a subsequent call to `setlocale()`). Therefore, if the purpose of
 61590 calling `setlocale()` is to save the value of the current international environment so it can be
 61591 changed and reset later, the return value should be copied to an array of `char` in the calling
 61592 program.

61593 There are three ways to set the international environment with `setlocale()`:

61594 `setlocale(category, string)`

61595 This usage sets a specific *category* in the international environment to a specific value
 61596 corresponding to the value of the *string*. A specific example is provided below:

61597 `setlocale(LC_ALL, "fr_FR.ISO-8859-1");`

61598 In this example, all categories of the international environment are set to the locale
61599 corresponding to the string "fr_FR.ISO-8859-1", or to the French language as spoken in
61600 France using the ISO/IEC 8859-1:1998 standard codeset.

61601 If the string does not correspond to a valid locale, *setlocale()* shall return a null pointer and
61602 the international environment is not changed. Otherwise, *setlocale()* shall return the name of
61603 the locale just set.

61604 *setlocale(category, "C")*

61605 The ISO C standard states that one locale must exist on all conforming implementations.
61606 The name of the locale is C and corresponds to a minimal international environment needed
61607 to support the C programming language.

61608 *setlocale(category, "")*

61609 This sets a specific category to an implementation-defined default. This corresponds to the
61610 value of the environment variables.

61611 FUTURE DIRECTIONS

61612 None.

61613 SEE ALSO

61614 *catopen()*, *exec*, *fprintf()*, *fscanf()*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*,
61615 *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *iswalnum()*, *iswalpha()*, *iswblank()*, *iswcntrl()*,
61616 *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*,
61617 *iswxdigit()*, *isxdigit()*, *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *newlocale()*, *nl_langinfo()*,
61618 *perror()*, *psiginfo()*, *strcoll()*, *strerror()*, *strfmon()*, *strsignal()*, *strtod()*, *strxfrm()*, *tolower()*,
61619 *toupper()*, *towlower()*, *towupper()*, *uselocale()*, *wcscoll()*, *wcstod()*, *wcstombs()*, *wcsxfrm()*, *wctomb()*

61620 XBD Chapter 7 (on page 135), Chapter 8 (on page 173), [<langinfo.h>](#), [<locale.h>](#)

61621 CHANGE HISTORY

61622 First released in Issue 3.

61623 Issue 5

61624 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

61625 Issue 6

61626 Extensions beyond the ISO C standard are marked.

61627 The normative text is updated to avoid use of the term "must" for application requirements.

61628 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/124 is applied, updating the
61629 DESCRIPTION to clarify the behavior of:

61630 `setlocale(LC_ALL, "");`

61631 Issue 7

61632 Functionality relating to the Threads option is moved to the Base.

61633 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0553 [302], XSH/TC1-2008/0554 [303],
61634 XSH/TC1-2008/0555 [302], XSH/TC1-2008/0556 [302], XSH/TC1-2008/0557 [302],
61635 XSH/TC1-2008/0558 [302], XSH/TC1-2008/0559 [302], XSH/TC1-2008/0560 [288],
61636 XSH/TC1-2008/0561 [302], XSH/TC1-2008/0562 [302], XSH/TC1-2008/0563 [302],
61637 XSH/TC1-2008/0564 [302], XSH/TC1-2008/0565 [302], XSH/TC1-2008/0566 [302],
61638 XSH/TC1-2008/0567 [288], XSH/TC1-2008/0568 [288], and XSH/TC1-2008/0569 [303] are
61639 applied.

61640
61641

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0321 [826], XSH/TC2-2008/0322 [826], and XSH/TC2-2008/0323 [596] are applied.

61642 **NAME**

61643 setlogmask ¶set the log priority mask

61644 **SYNOPSIS**

61645 XSI #include <syslog.h>

61646 int setlogmask(int *maskpri*);61647 **DESCRIPTION**61648 Refer to *closelog()*.

61649 **NAME**
61650 setnetent †'network database function

61651 **SYNOPSIS**
61652 #include <netdb.h>
61653 void setnetent(int *stayopen*);

61654 **DESCRIPTION**
61655 Refer to *endnetent()*.

61656 **NAME**

61657 setpgid — set process group ID for job control

61658 **SYNOPSIS**

61659 #include <unistd.h>

61660 int setpgid(pid_t pid, pid_t pgid);

61661 **DESCRIPTION**61662 The *setpgid()* function shall either join an existing process group or create a new process group
61663 within the session of the calling process.

61664 The process group ID of a session leader shall not change.

61665 Upon successful completion, the process group ID of the process with a process ID that matches
61666 *pid* shall be set to *pgid*.61667 As a special case, if *pid* is 0, the process ID of the calling process shall be used. Also, if *pgid* is 0,
61668 the process ID of the indicated process shall be used.61669 **RETURN VALUE**61670 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*
61671 shall be set to indicate the error.61672 **ERRORS**61673 The *setpgid()* function shall fail if:61674 [EACCES] The value of the *pid* argument matches the process ID of a child process of the
61675 calling process and the child process has successfully executed one of the *exec*
61676 functions.61677 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by
61678 the implementation.61679 [EPERM] The process indicated by the *pid* argument is a session leader.61680 [EPERM] The value of the *pid* argument matches the process ID of a child process of the
61681 calling process and the child process is not in the same session as the calling
61682 process.61683 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of
61684 the process indicated by the *pid* argument and there is no process with a
61685 process group ID that matches the value of the *pgid* argument in the same
61686 session as the calling process.61687 [ESRCH] The value of the *pid* argument does not match the process ID of the calling
61688 process or of a child process of the calling process.61689 **EXAMPLES**

61690 None.

61691 **APPLICATION USAGE**

61692 None.

61693 **RATIONALE**61694 The *setpgid()* function shall group processes together for the purpose of signaling, placement in
61695 foreground or background, and other job control actions.61696 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed
61697 the specified new process group to assume any value. This presents certain security problems
61698 and is more flexible than necessary to support job control.

61699 To provide tighter security, *setpgid()* only allows the calling process to join a process group
 61700 already in use inside its session or create a new process group whose process group ID was
 61701 equal to its process ID.

61702 When a job control shell spawns a new job, the processes in the job must be placed into a new
 61703 process group via *setpgid()*. There are two timing constraints involved in this action:

- 61704 1. The new process must be placed in the new process group before the appropriate
 61705 program is launched via one of the *exec* functions.
- 61706 2. The new process must be placed in the new process group before the shell can correctly
 61707 send signals to the new process group.

61708 To address these constraints, the following actions are performed. The new processes call
 61709 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first
 61710 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of
 61711 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that
 61712 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization
 61713 property was considered, but it was decided instead to merely allow the parent shell process to
 61714 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now
 61715 satisfied by having both the parent shell and the child attempt to adjust the process group of the
 61716 child process; it does not matter which succeeds first.

61717 Since it would be confusing to an application to have its process group change after it began
 61718 executing (that is, after *exec*), and because the child process would already have adjusted its
 61719 process group before this, the [EACCES] error was added to disallow this.

61720 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original
 61721 process group (the one in effect when the job control shell was executed). A job control shell
 61722 does this before returning control back to its parent when it is terminating or suspending itself as
 61723 a way of restoring its job control “state” back to what its parent would expect. (Note that the
 61724 original process group of the job control shell typically matches the process group of its parent,
 61725 but this is not necessarily always the case.)

61726 FUTURE DIRECTIONS

61727 None.

61728 SEE ALSO

61729 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*

61730 XBD [<sys/types.h>](#), [<unistd.h>](#)

61731 CHANGE HISTORY

61732 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61733 Issue 6

61734 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

61735 The following new requirements on POSIX implementations derive from alignment with the
 61736 Single UNIX Specification:

61737 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 61738 required for conforming implementations of previous POSIX specifications, it was not
 61739 required for UNIX applications.

61740 The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be
 61741 defined in this version. This is a FIPS requirement.

61742
61743
61744
61745

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the process ID of the indicated process shall be used”. This change reverts the wording to as in the ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

61746 **NAME**

61747 setpgrp — set the process group ID

61748 **SYNOPSIS**

```
61749 OB XSI #include <unistd.h>  
61750 pid_t setpgrp(void);
```

61751 **DESCRIPTION**

61752 If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the
61753 calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then the
61754 new session has no controlling terminal.

61755 The *setpgrp()* function has no effect when the calling process is a session leader.

61756 **RETURN VALUE**

61757 Upon completion, *setpgrp()* shall return the process group ID.

61758 **ERRORS**

61759 No errors are defined.

61760 **EXAMPLES**

61761 None.

61762 **APPLICATION USAGE**

61763 It is unspecified whether this function behaves as *setpgid(0,0)* or *setsid()* unless the process is
61764 already a session leader. Therefore, applications are encouraged to use *setpgid()* or *setsid()* as
61765 appropriate.

61766 **RATIONALE**

61767 None.

61768 **FUTURE DIRECTIONS**

61769 The *setpgrp()* function may be removed in a future version.

61770 **SEE ALSO**

61771 *exec*, *fork()*, *getpid()*, *getsid()*, *kill()*, *setpgid()*, *setsid()*

61772 XBD <unistd.h>

61773 **CHANGE HISTORY**

61774 First released in Issue 4, Version 2.

61775 **Issue 5**

61776 Moved from X/OPEN UNIX extension to BASE.

61777 **Issue 7**

61778 The *setpgrp()* function is marked obsolescent.

61779 **NAME**

61780 setpriority ‡set the nice value

61781 **SYNOPSIS**

61782 XSI #include <sys/resource.h>

61783 int setpriority(int *which*, id_t *who*, int *nice*);61784 **DESCRIPTION**61785 Refer to [getpriority\(\)](#).

61786 **NAME**
61787 setprotoent — network protocol database functions

61788 **SYNOPSIS**
61789 #include <netdb.h>

61790 void setprotoent(int *stayopen*);

61791 **DESCRIPTION**
61792 Refer to *endprotoent()*.

61793 **NAME**

61794 setpwent ‡'user database function

61795 **SYNOPSIS**

61796 XSI #include <pwd.h>

61797 void setpwent(void);

61798 **DESCRIPTION**61799 Refer to *endpwent()*.

61800 **NAME**

61801 setregid — set real and effective group IDs

61802 **SYNOPSIS**

```
61803 XSI #include <unistd.h>
61804 int setregid(gid_t rgid, gid_t egid);
```

61805 **DESCRIPTION**61806 The *setregid()* function shall set the real and effective group IDs of the calling process.61807 If *rgid* is -1 , the real group ID shall not be changed; if *egid* is -1 , the effective group ID shall not
61808 be changed.

61809 The real and effective group IDs may be set to different values in the same call.

61810 Only a process with appropriate privileges can set the real group ID and the effective group ID
61811 to any valid value.61812 A non-privileged process can set either the real group ID to the saved set-group-ID from one of
61813 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group
61814 ID.61815 If the real group ID is being set (*rgid* is not -1), or the effective group ID is being set to a value
61816 not equal to the real group ID, then the saved set-group-ID of the current process shall be set
61817 equal to the new effective group ID.

61818 Any supplementary group IDs of the calling process remain unchanged.

61819 **RETURN VALUE**61820 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
61821 indicate the error, and neither of the group IDs are changed.61822 **ERRORS**61823 The *setregid()* function shall fail if:61824 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.61825 [EPERM] The process does not have appropriate privileges and a change other than
61826 changing the real group ID to the saved set-group-ID, or changing the
61827 effective group ID to the real group ID or the saved set-group-ID, was
61828 requested.61829 **EXAMPLES**

61830 None.

61831 **APPLICATION USAGE**61832 If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can
61833 only set its effective group ID back to the previous value if *rgid* was -1 in the *setregid()* call, since
61834 the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the
61835 *setregid()* call, then the saved set-group-ID will also have been changed to the real user ID.61836 **RATIONALE**61837 Earlier versions of this standard did not specify whether the saved set-group-ID was affected by
61838 *setregid()* calls. This version specifies common existing practice that constitutes an important
61839 security feature. The ability to set both the effective group ID and saved set-group-ID to be the
61840 same as the real group ID means that any security weakness in code that is executed after that
61841 point cannot result in malicious code being executed with the previous effective group ID.
61842 Privileged applications could already do this using just *setgid()*, but for non-privileged

61843 applications the only standard method available is to use this feature of *setregid()*.

61844 **FUTURE DIRECTIONS**

61845 None.

61846 **SEE ALSO**

61847 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*

61848 XBD <[unistd.h](#)>

61849 **CHANGE HISTORY**

61850 First released in Issue 4, Version 2.

61851 **Issue 5**

61852 Moved from X/OPEN UNIX extension to BASE.

61853 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the
61854 *exec* family of functions, not just *execve()*.

61855 **Issue 7**

61856 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-
61857 group-ID to be the same as the real group ID.

61858 **NAME**

61859 setreuid — set real and effective user IDs

61860 **SYNOPSIS**

```
61861 XSI      #include <unistd.h>
61862         int setreuid(uid_t ruid, uid_t euid);
```

61863 **DESCRIPTION**

61864 The *setreuid()* function shall set the real and effective user IDs of the current process to the
 61865 values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is *-1*, the corresponding effective
 61866 or real user ID of the current process shall be left unchanged.

61867 A process with appropriate privileges can set either ID to any value. An unprivileged process
 61868 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or
 61869 saved user ID of the process.

61870 If the real user ID is being set (*ruid* is not *-1*), or the effective user ID is being set to a value not
 61871 equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to
 61872 the new effective user ID.

61873 It is unspecified whether a process without appropriate privileges is permitted to change the real
 61874 user ID to match the current effective user ID or saved set-user-ID of the process.

61875 **RETURN VALUE**

61876 Upon successful completion, 0 shall be returned. Otherwise, *-1* shall be returned and *errno* set to
 61877 indicate the error.

61878 **ERRORS**

61879 The *setreuid()* function shall fail if:

- | | | |
|-------|----------|--|
| 61880 | [EINVAL] | The value of the <i>ruid</i> or <i>euid</i> argument is invalid or out-of-range. |
| 61881 | [EPERM] | The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation. |

61885 **EXAMPLES**61886 **Setting the Effective User ID to the Real User ID**

61887 The following example sets the effective user ID of the calling process to the real user ID, so that
 61888 files created later will be owned by the current user. It also sets the saved set-user-ID to the real
 61889 user ID, so any future attempt to set the effective user ID back to its previous value will fail.

```
61890 #include <unistd.h>
61891 #include <sys/types.h>
61892 ...
61893 setreuid(getuid(), getuid());
61894 ...
```

61895 **APPLICATION USAGE**

61896 None.

61897 RATIONALE

61898 Earlier versions of this standard did not specify whether the saved set-user-ID was affected by
61899 *setreuid()* calls. This version specifies common existing practice that constitutes an important
61900 security feature. The ability to set both the effective user ID and saved set-user-ID to be the same
61901 as the real user ID means that any security weakness in code that is executed after that point
61902 cannot result in malicious code being executed with the previous effective user ID. Privileged
61903 applications could already do this using just *setuid()*, but for non-privileged applications the
61904 only standard method available is to use this feature of *setreuid()*.

61905 FUTURE DIRECTIONS

61906 None.

61907 SEE ALSO

61908 *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*

61909 XBD <[unistd.h](#)>

61910 CHANGE HISTORY

61911 First released in Issue 4, Version 2.

61912 Issue 5

61913 Moved from X/OPEN UNIX extension to BASE.

61914 Issue 7

61915 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved
61916 set-user-ID to be the same as the real user ID.

61917 **NAME**

61918 setrlimit — control maximum resource consumption

61919 **SYNOPSIS**

61920 XSI #include <sys/resource.h>

61921 int setrlimit(int resource, const struct rlimit *rlp);

61922 **DESCRIPTION**

61923 Refer to [getrlimit\(\)](#).

61924 **NAME**
61925 setservent ‡network services database functions

61926 **SYNOPSIS**
61927 #include <netdb.h>
61928 void setservent(int stayopen);

61929 **DESCRIPTION**
61930 Refer to *endservent()*.

61931 **NAME**

61932 setsid — create session and set process group ID

61933 **SYNOPSIS**

```
61934 #include <unistd.h>
61935 pid_t setsid(void);
```

61936 **DESCRIPTION**

61937 The *setsid()* function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process. The calling process shall be the only process in the new process group and the only process in the new session.

61943 **RETURN VALUE**

61944 Upon successful completion, *setsid()* shall return the value of the new process group ID of the calling process. Otherwise, it shall return `-1` and set *errno* to indicate the error.

61946 **ERRORS**

61947 The *setsid()* function shall fail if:

61948	[EPERM]	The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.
61949		
61950		

61951 **EXAMPLES**

61952 None.

61953 **APPLICATION USAGE**

61954 None.

61955 **RATIONALE**

61956 The *setsid()* function is similar to the *setpgrp()* function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgrp()*; only one process group may be part of a login session.

61959 Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of POSIX.1-2017 adds the concept of a session that is created via *setsid()*. The *setsid()* function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid()* function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function guarantees that the process ID of the new process does not match any existing process group ID.

61968 **FUTURE DIRECTIONS**

61969 None.

61970 **SEE ALSO**

61971 [getsid\(\)](#), [setpgid\(\)](#), [setpgrp\(\)](#)

61972 XBD [<sys/types.h>](#), [<unistd.h>](#)

61973 **CHANGE HISTORY**

61974 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

61975 **Issue 6**

61976 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

61977 The following new requirements on POSIX implementations derive from alignment with the
61978 Single UNIX Specification:

61979 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
61980 required for conforming implementations of previous POSIX specifications, it was not
61981 required for UNIX applications.

61982 **Issue 7**

61983 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0570 [421] is applied.

61984 **NAME**

61985 setsockopt ¶set the socket options

61986 **SYNOPSIS**

61987 #include <sys/socket.h>

61988 int setsockopt(int *socket*, int *level*, int *option_name*,
61989 const void **option_value*, socklen_t *option_len*);61990 **DESCRIPTION**61991 The *setsockopt()* function shall set the option specified by the *option_name* argument, at the
61992 protocol level specified by the *level* argument, to the value pointed to by the *option_value*
61993 argument for the socket associated with the file descriptor specified by the *socket* argument.61994 The *level* argument specifies the protocol level at which the option resides. To set options at the
61995 socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply
61996 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate
61997 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP
61998 as defined in the <netinet/in.h> header.61999 The *option_name* argument specifies a single option to set. It can be one of the socket-level
62000 options defined in <sys/socket.h> and described in Section 2.10.16 (on page 528). If *option_name*
62001 is equal to SO_RCVTIMEO or SO_SNDTIMEO and the implementation supports setting the
62002 option, it is unspecified whether the **struct timeval** pointed to by *option_value* is stored as
62003 provided by this function or is rounded up to align with the resolution of the clock being used. If
62004 *setsockopt()* is called with *option_name* equal to SO_ACCEPTCONN, SO_ERROR, or SO_TYPE,
62005 the behavior is unspecified.62006 **RETURN VALUE**62007 Upon successful completion, *setsockopt()* shall return 0. Otherwise, -1 shall be returned and
62008 *errno* set to indicate the error.62009 **ERRORS**62010 The *setsockopt()* function shall fail if:62011 [EBADF] The *socket* argument is not a valid file descriptor.62012 [EDOM] The send and receive timeout values are too big to fit into the timeout fields in
62013 the socket structure.62014 [EINVAL] The specified option is invalid at the specified socket level or the socket has
62015 been shut down.62016 [EISCONN] The socket is already connected, and a specified option cannot be set while the
62017 socket is connected.

62018 [ENOPROTOOPT]

62019 The option is not supported by the protocol.

62020 [ENOTSOCK] The *socket* argument does not refer to a socket.62021 The *setsockopt()* function may fail if:

62022 [ENOMEM] There was insufficient memory available for the operation to complete.

62023 [ENOBUFS] Insufficient resources are available in the system to complete the call.

62024 EXAMPLES

62025 None.

62026 APPLICATION USAGE

62027 The *setsockopt()* function provides an application program with the means to control socket
62028 behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts,
62029 or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options
62030 available to *setsockopt()*.

62031 Options may exist at multiple protocol levels. The `SO_` options are always present at the
62032 uppermost socket level.

62033 RATIONALE

62034 None.

62035 FUTURE DIRECTIONS

62036 None.

62037 SEE ALSO

62038 [Section 2.10](#) (on page 523), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*

62039 XBD `<netinet/in.h>`, `<sys/socket.h>`

62040 CHANGE HISTORY

62041 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

62042 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/125 is applied, updating the `SO_LINGER`
62043 option in the DESCRIPTION to refer to the calling thread rather than the process.

62044 Issue 7

62045 Austin Group Interpretation 1003.1-2001 #158 is applied, removing text relating to socket options
62046 that is now in [Section 2.10.16](#) (on page 528).

62047 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0571 [369] is applied.

62048 **NAME**

62049 setstate ‡switch pseudo-random number generator state arrays

62050 **SYNOPSIS**

```
62051 XSI       #include <stdlib.h>  
62052       char *setstate(char *state);
```

62053 **DESCRIPTION**

62054 Refer to *initstate()*.

62055 **NAME**

62056 setuid †'set user ID

62057 **SYNOPSIS**

62058 #include <unistd.h>

62059 int setuid(uid_t uid);

62060 **DESCRIPTION**62061 If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and
62062 the saved set-user-ID of the calling process to *uid*.62063 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the
62064 saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-
62065 user-ID shall remain unchanged.62066 The *setuid()* function shall not affect the supplementary group list in any way.62067 **RETURN VALUE**62068 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
62069 indicate the error.62070 **ERRORS**62071 The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more
62072 of the following are true:62073 [EINVAL] The value of the *uid* argument is invalid and not supported by the
62074 implementation.62075 [EPERM] The process does not have appropriate privileges and *uid* does not match the
62076 real user ID or the saved set-user-ID.62077 **EXAMPLES**

62078 None.

62079 **APPLICATION USAGE**

62080 None.

62081 **RATIONALE**62082 The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged
62083 processes reflect the behavior of different historical implementations. For portability, it is
62084 recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions
62085 instead.62086 The saved set-user-ID capability allows a program to regain the effective user ID established at
62087 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the
62088 effective group ID established at the last *exec* call. These capabilities are derived from System V.
62089 Without them, a program might have to run as superuser in order to perform the same
62090 functions, because superuser can write on the user's files. This is a problem because such a
62091 program can write on any user's files, and so must be carefully written to emulate the
62092 permissions of the calling process properly. In System V, these capabilities have traditionally
62093 been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The
62094 fact that the behavior of those functions was different for privileged processes made them
62095 difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently
62096 for privileged and unprivileged users. When the caller had appropriate privileges, the function
62097 set the real user ID, effective user ID, and saved set-user ID of the calling process on
62098 implementations that supported it. When the caller did not have appropriate privileges, the
62099 function set only the effective user ID, subject to permission checks. The former use is generally
62100 needed for utilities like *login* and *su*, which are not conforming applications and thus outside the

62101 scope of POSIX.1-2017. These utilities wish to change the user ID irrevocably to a new value,
 62102 generally that of an unprivileged user. The latter use is needed for conforming applications that
 62103 are installed with the set-user-ID bit and need to perform operations using the real user ID.

62104 POSIX.1-2017 augments the latter functionality with a mandatory feature named
 62105 `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user
 62106 ID back and forth between the values of its *exec*-time real user ID and effective user ID.
 62107 Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this
 62108 feature to work properly when it happened to be executed with (implementation-defined)
 62109 appropriate privileges. Furthermore, the application did not even have a means to tell whether it
 62110 had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as
 62111 evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-2017.
 62112 However, there are implementors who have been reluctant to support it given the limitation
 62113 described above.

62114 The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which
 62115 always sets both the real and effective user IDs, like *setuid()* in POSIX.1-2017 for privileged
 62116 users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in POSIX.1-2017
 62117 for non-privileged users). This separation of functionality into distinct functions seems desirable.
 62118 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of
 62119 switching the effective user ID back and forth via *setreuid()*, which permits reversing the real
 62120 and effective user IDs. This model seems less desirable than the saved set-user-ID because the
 62121 real user ID changes as a side-effect. The current 4.4BSD includes saved effective IDs and uses
 62122 them for *seteuid()* and *setegid()* as described above. The *setreuid()* and *setregid()* functions will be
 62123 deprecated or removed.

62124 The solution here is:

62125 Require that all implementations support the functionality of the saved set-user-ID, which
 62126 is set by the *exec* functions and by privileged calls to *setuid()*.

62127 Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for
 62128 non-privileged and privileged processes.

62129 Historical systems have provided two mechanisms for a set-user-ID process to change its
 62130 effective user ID to be the same as its real user ID in such a way that it could return to the
 62131 original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID,
 62132 or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs.
 62133 The changes included in POSIX.1-2017 provide a new mechanism using *seteuid()* in conjunction
 62134 with a saved set-user-ID. Thus, all implementations with the new *seteuid()* mechanism will have
 62135 a saved set-user-ID for each process, and most of the behavior controlled by
 62136 `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined.
 62137 The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally
 62138 be required to maintain compatibility with the older mechanisms previously supported by their
 62139 systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS`
 62140 behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID
 62141 allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the
 62142 saved set-user-ID unmodified, the process would then have an effective user ID equal to the
 62143 original real user ID, and both real and saved set-user-ID would be equal to the original effective
 62144 user ID. In that state, the real user would be unable to kill the process, even though the effective
 62145 user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS`
 62146 was used. This is obviously not acceptable. The alternative choice, which is used in at least one
 62147 implementation, is to change the saved set-user-ID to the effective user ID during most calls to
 62148 *setreuid()*. The standard developers considered that alternative to be less correct than the
 62149 retention of the old behavior of *kill()* in such systems. Current conforming applications shall

62150 accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to
62151 check the saved set-user-ID rather than the effective user ID.

62152 **FUTURE DIRECTIONS**

62153 None.

62154 **SEE ALSO**

62155 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*

62156 XBD [<sys/types.h>](#), [<unistd.h>](#)

62157 **CHANGE HISTORY**

62158 First released in Issue 1. Derived from Issue 1 of the SVID.

62159 **Issue 6**

62160 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

62161 The following new requirements on POSIX implementations derive from alignment with the
62162 Single UNIX Specification:

62163 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
62164 required for conforming implementations of previous POSIX specifications, it was not
62165 required for UNIX applications.

62166 The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS
62167 requirement.

62168 The following changes were made to align with the IEEE P1003.1a draft standard:

62169 The effects of *setuid()* in processes without appropriate privileges are changed.

62170 A requirement that the supplementary group list is not affected is added.

62171 **NAME**

62172 setutxent — reset the user accounting database to the first entry

62173 **SYNOPSIS**

```
62174 XSI       #include <utmpx.h>  
62175           void setutxent(void);
```

62176 **DESCRIPTION**

62177 Refer to *endutxent()*.

62178 **NAME**

62179 setvbuf — assign buffering to a stream

62180 **SYNOPSIS**

62181 #include <stdio.h>

62182 int setvbuf(FILE *restrict stream, char *restrict buf, int type,
62183 size_t size);62184 **DESCRIPTION**62185 CX The functionality described on this reference page is aligned with the ISO C standard. Any
62186 conflict between the requirements described here and the ISO C standard is unintentional. This
62187 volume of POSIX.1-2017 defers to the ISO C standard.62188 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an
62189 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is
62190 performed on the stream. The argument *type* determines how *stream* shall be buffered, as
62191 follows:

62192 {_IOFBF} shall cause input/output to be fully buffered.

62193 {_IOLBF} shall cause input/output to be line buffered.

62194 {_IONBF} shall cause input/output to be unbuffered.

62195 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by
62196 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the
62197 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are
62198 unspecified.62199 For information about streams, see [Section 2.5](#) (on page 495).62200 **RETURN VALUE**62201 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value
62202 CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to
62203 indicate the error.62204 **ERRORS**62205 The *setvbuf()* function may fail if:62206 CX [EBADF] The file descriptor underlying *stream* is not valid.62207 **EXAMPLES**

62208 None.

62209 **APPLICATION USAGE**62210 A common source of error is allocating buffer space as an “automatic” variable in a code block,
62211 and then failing to close the stream in the same block.62212 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are
62213 used for the buffer area.62214 Applications should note that many implementations only provide line buffering on input from
62215 terminal devices.62216 **RATIONALE**

62217 None.

62218 **FUTURE DIRECTIONS**

62219 None.

62220 **SEE ALSO**62221 [Section 2.5](#) (on page 495), [fopen\(\)](#), [setbuf\(\)](#)62222 XBD [<stdio.h>](#)62223 **CHANGE HISTORY**

62224 First released in Issue 1. Derived from Issue 1 of the SVID.

62225 **Issue 6**

62226 Extensions beyond the ISO C standard are marked.

62227 The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

62228 **NAME**62229 shm_open — open a shared memory object (**REALTIME**)62230 **SYNOPSIS**

```
62231 SHM    #include <sys/mman.h>
62232      int shm_open(const char *name, int oflag, mode_t mode);
```

62233 **DESCRIPTION**

62234 The *shm_open()* function shall establish a connection between a shared memory object and a file
 62235 descriptor. It shall create an open file description that refers to the shared memory object and a
 62236 file descriptor that refers to that open file description. The file descriptor shall be allocated as
 62237 described in [Section 2.14](#) (on page 549), and can be used by other functions to refer to that shared
 62238 memory object. The *name* argument points to a string naming a shared memory object. It is
 62239 unspecified whether the name appears in the file system and is visible to other functions that
 62240 take pathnames as arguments. The *name* argument conforms to the construction rules for a
 62241 pathname, except that the interpretation of <slash> characters other than the leading <slash>
 62242 character in *name* is implementation-defined, and that the length limits for the *name* argument
 62243 are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and
 62244 {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *shm_open()*
 62245 with the same value of *name* refer to the same shared memory object, as long as that name has
 62246 not been removed. If *name* does not begin with the <slash> character, the effect is
 62247 implementation-defined.

62248 If successful, *shm_open()* shall return a file descriptor for the shared memory object. The open
 62249 file description is new, and therefore the file descriptor does not share it with any other
 62250 processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag
 62251 associated with the new file descriptor is set.

62252 The file status flags and file access modes of the open file description are according to the value
 62253 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the
 62254 <fcntl.h> header. Applications specify exactly one of the first two values (access modes) below
 62255 in the value of *oflag*:

62256 O_RDONLY Open for read access only.

62257 O_RDWR Open for read or write access.

62258 Any combination of the remaining flags may be specified in the value of *oflag*:

62259 O_CREAT If the shared memory object exists, this flag has no effect, except as noted
 62260 under O_EXCL below. Otherwise, the shared memory object is created. The
 62261 user ID of the shared memory object shall be set to the effective user ID of the
 62262 process. The group ID of the shared memory object shall be set to the effective
 62263 group ID of the process; however, if the *name* argument is visible in the file
 62264 system, the group ID may be set to the group ID of the containing directory.
 62265 The permission bits of the shared memory object shall be set to the value of
 62266 the *mode* argument except those set in the file mode creation mask of the
 62267 process. When bits in *mode* other than the file permission bits are set, the effect
 62268 is unspecified. The *mode* argument does not affect whether the shared memory
 62269 object is opened for reading, for writing, or for both. The shared memory
 62270 object has a size of zero.

62271 O_EXCL If O_EXCL and O_CREAT are set, *shm_open()* fails if the shared memory
 62272 object exists. The check for the existence of the shared memory object and the
 62273 creation of the object if it does not exist is atomic with respect to other

62274 processes executing *shm_open()* naming the same shared memory object with
 62275 O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the
 62276 result is undefined.

62277 O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR, the
 62278 object shall be truncated to zero length and the mode and owner shall be
 62279 unchanged by this function call. The result of using O_TRUNC with
 62280 O_RDONLY is undefined.

62281 When a shared memory object is created, the state of the shared memory object, including all
 62282 data associated with the shared memory object, persists until the shared memory object is
 62283 unlinked and all other references are gone. It is unspecified whether the name and shared
 62284 memory object state remain valid after a system reboot.

62285 RETURN VALUE

62286 Upon successful completion, the *shm_open()* function shall return a non-negative integer
 62287 representing the file descriptor. Otherwise, it shall return -1 and set *errno* to indicate the error.

62288 ERRORS

62289 The *shm_open()* function shall fail if:

62290 [EACCES] The shared memory object exists and the permissions specified by *oflag* are
 62291 denied, or the shared memory object does not exist and permission to create
 62292 the shared memory object is denied, or O_TRUNC is specified and write
 62293 permission is denied.

62294 [EEXIST] O_CREAT and O_EXCL are set and the named shared memory object already
 62295 exists.

62296 [EINTR] The *shm_open()* operation was interrupted by a signal.

62297 [EINVAL] The *shm_open()* operation is not supported for the given name.

62298 [EMFILE] All file descriptors available to the process are currently open.

62299 [ENFILE] Too many shared memory objects are currently open in the system.

62300 [ENOENT] O_CREAT is not set and the named shared memory object does not exist.

62301 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

62302 The *shm_open()* function may fail if:

62303 [ENAMETOOLONG]

62304 The length of the *name* argument exceeds $\{_POSIX_PATH_MAX\}$ on systems
 62305 XSI that do not support the XSI option or exceeds $\{_XOPEN_PATH_MAX\}$ on XSI
 62306 systems, or has a pathname component that is longer than
 62307 XSI $\{_POSIX_NAME_MAX\}$ on systems that do not support the XSI option or
 62308 longer than $\{_XOPEN_NAME_MAX\}$ on XSI systems.

62309 **EXAMPLES**62310 **Creating and Mapping a Shared Memory Object**

62311 The following code segment demonstrates the use of *shm_open()* to create a shared memory
62312 object which is then sized using *ftruncate()* before being mapped into the process address space
62313 using *mmap()*:

```
62314 #include <unistd.h>
62315 #include <sys/mman.h>
62316 ...
62317 #define MAX_LEN 10000
62318 struct region {          /* Defines "structure" of shared memory */
62319     int len;
62320     char buf[MAX_LEN];
62321 };
62322 struct region *rptr;
62323 int fd;
62324 /* Create shared memory object and set its size */
62325 fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
62326 if (fd == -1)
62327     /* Handle error */;
62328 if (ftruncate(fd, sizeof(struct region)) == -1)
62329     /* Handle error */;
62330 /* Map shared memory object */
62331 rptr = mmap(NULL, sizeof(struct region),
62332            PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
62333 if (rptr == MAP_FAILED)
62334     /* Handle error */;
62335 /* Now we can refer to mapped region using fields of rptr;
62336    for example, rptr->len */
62337 ...
```

62338 **APPLICATION USAGE**

62339 None.

62340 **RATIONALE**

62341 When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a
62342 descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared
62343 Memory Objects option is supported, the *shm_open()* function shall obtain a descriptor to the
62344 shared memory object to be mapped.

62345 There is ample precedent for having a file descriptor represent several types of objects. In the
62346 POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory.
62347 Many implementations simply have an operations vector, which is indexed by the file descriptor
62348 type and does very different operations. Note that in some cases the file descriptor passed to
62349 generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned
62350 by alternate functions, such as *pipe()*. The latter technique is used by *shm_open()*.

62351 Note that such shared memory objects can actually be implemented as mapped files. In both
62352 cases, the size can be set after the open using *ftruncate()*. The *shm_open()* function itself does not

62353 create a shared object of a specified size because this would duplicate an extant function that set
62354 the size of an object referenced by a file descriptor.

62355 On implementations where memory objects are implemented using the existing file system, the
62356 *shm_open()* function may be implemented using a macro that invokes *open()*, and the
62357 *shm_unlink()* function may be implemented using a macro that invokes *unlink()*.

62358 For implementations without a permanent file system, the definition of the name of the memory
62359 objects is allowed not to survive a system reboot. Note that this allows systems with a
62360 permanent file system to implement memory objects as data structures internal to the
62361 implementation as well.

62362 On implementations that choose to implement memory objects using memory directly, a
62363 *shm_open()* followed by an *truncate()* and *close()* can be used to preallocate a shared memory
62364 area and to set the size of that preallocation. This may be necessary for systems without virtual
62365 memory hardware support in order to ensure that the memory is contiguous.

62366 The set of valid open flags to *shm_open()* was restricted to *O_RDONLY*, *O_RDWR*, *O_CREAT*,
62367 and *O_TRUNC* because these could be easily implemented on most memory mapping systems.
62368 This volume of POSIX.1-2017 is silent on the results if the implementation cannot supply the
62369 requested file access because of implementation-defined reasons, including hardware ones.

62370 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the
62371 implementation cannot complete a request.

62372 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the
62373 implementation cannot comply with a requested mode because it conflicts with another
62374 requested mode. An example might be that an application desires to open a memory object two
62375 times, mapping different areas with different access modes. If the implementation cannot map a
62376 single area into a process space in two places, which would be required if different access modes
62377 were required for the two areas, then the implementation may inform the application at the time
62378 of the second open.

62379 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the
62380 implementation cannot comply with a requested mode at all. An example would be that the
62381 hardware of the implementation cannot support write-only shared memory areas.

62382 On all implementations, it may be desirable to restrict the location of the memory objects to
62383 specific file systems for performance (such as a RAM disk) or implementation-defined reasons
62384 (shared memory supported directly only on certain file systems). The *shm_open()* function may
62385 be used to enforce these restrictions. There are a number of methods available to the application
62386 to determine an appropriate name of the file or the location of an appropriate directory. One way
62387 is from the environment via *getenv()*. Another would be from a configuration file.

62388 This volume of POSIX.1-2017 specifies that memory objects have initial contents of zero when
62389 created. This is consistent with current behavior for both files and newly allocated memory. For
62390 those implementations that use physical memory, it would be possible that such
62391 implementations could simply use available memory and give it to the process uninitialized.
62392 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,
62393 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security
62394 reasons. Thus, initializing memory objects to zero is required.

62395 **FUTURE DIRECTIONS**

62396 A future version might require the *shm_open()* and *shm_unlink()* functions to have semantics
62397 similar to normal file system operations.

62398 **SEE ALSO**

62399 [Section 2.14](#) (on page 549), [close\(\)](#), [dup\(\)](#), [exec](#), [fcntl\(\)](#), [mmap\(\)](#), [shmat\(\)](#), [shmctl\(\)](#), [shmdt\(\)](#),
62400 [shm_unlink\(\)](#), [umask\(\)](#)

62401 XBD [<fcntl.h>](#), [<sys/mman.h>](#)

62402 **CHANGE HISTORY**

62403 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

62404 **Issue 6**

62405 The `shm_open()` function is marked as part of the Shared Memory Objects option.

62406 The [ENOSYS] error condition has been removed as stubs need not be provided if an
62407 implementation does not support the Shared Memory Objects option.

62408 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/126 is applied, adding the example to the
62409 EXAMPLES section.

62410 **Issue 7**

62411 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and
62412 changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

62413 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

62414 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the
62415 user ID and group ID of the shared memory object.

62416 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0324 [835], XSH/TC2-2008/0325 [835],
62417 and XSH/TC2-2008/0326 [835] are applied.

62418 **NAME**62419 shm_unlink — remove a shared memory object (**REALTIME**)62420 **SYNOPSIS**

```
62421 SHM    #include <sys/mman.h>
62422        int shm_unlink(const char *name);
```

62423 **DESCRIPTION**

62424 The *shm_unlink()* function shall remove the name of the shared memory object named by the
62425 string pointed to by *name*.

62426 If one or more references to the shared memory object exist when the object is unlinked, the
62427 name shall be removed before *shm_unlink()* returns, but the removal of the memory object
62428 contents shall be postponed until all open and map references to the shared memory object have
62429 been removed.

62430 Even if the object continues to exist after the last *shm_unlink()*, reuse of the name shall
62431 subsequently cause *shm_open()* to behave as if no shared memory object of this name exists (that
62432 is, *shm_open()* will fail if *O_CREAT* is not set, or will create a new shared memory object if
62433 *O_CREAT* is set).

62434 **RETURN VALUE**

62435 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be
62436 returned and *errno* set to indicate the error. If -1 is returned, the named shared memory object
62437 shall not be changed by this function call.

62438 **ERRORS**

62439 The *shm_unlink()* function shall fail if:

62440 [EACCES] Permission is denied to unlink the named shared memory object.

62441 [ENOENT] The named shared memory object does not exist.

62442 The *shm_unlink()* function may fail if:

62443 [ENAMETOOLONG]

62444 The length of the *name* argument exceeds $\{_POSIX_PATH_MAX\}$ on systems
62445 XSI that do not support the XSI option or exceeds $\{_XOPEN_PATH_MAX\}$ on XSI
62446 systems, or has a pathname component that is longer than
62447 XSI $\{_POSIX_NAME_MAX\}$ on systems that do not support the XSI option or
62448 longer than $\{_XOPEN_NAME_MAX\}$ on XSI systems. A call to *shm_unlink()*
62449 with a *name* argument that contains the same shared memory object name as
62450 was previously used in a successful *shm_open()* call shall not give an
62451 [ENAMETOOLONG] error.

62452 **EXAMPLES**

62453 None.

62454 **APPLICATION USAGE**

62455 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual
62456 fashion. Names of memory objects that were allocated with *shm_open()* are deleted with
62457 *shm_unlink()*. Note that the actual memory object is not destroyed until the last close and
62458 unmap on it have occurred if it was already in use.

62459 **RATIONALE**

62460 None.

62461 **FUTURE DIRECTIONS**62462 A future version might require the *shm_open()* and *shm_unlink()* functions to have semantics
62463 similar to normal file system operations.62464 **SEE ALSO**62465 *close()*, *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_open()*62466 XBD <[sys/mman.h](#)>62467 **CHANGE HISTORY**

62468 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

62469 **Issue 6**62470 The *shm_unlink()* function is marked as part of the Shared Memory Objects option.62471 In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink()*
62472 will not attach to the old shared memory object.62473 The [ENOSYS] error condition has been removed as stubs need not be provided if an
62474 implementation does not support the Shared Memory Objects option.62475 **Issue 7**62476 Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a
62477 ``shall fail'' to a ``may fail'' error.

62478 Austin Group Interpretation 1003.1-2001 #141 is applied, adding FUTURE DIRECTIONS.

62479 **NAME**

62480 shmat — XSI shared memory attach operation

62481 **SYNOPSIS**

```
62482 XSI #include <sys/shm.h>
62483 void *shmat(int shmid, const void *shmaddr, int shmflg);
```

62484 **DESCRIPTION**

62485 The *shmat()* function operates on XSI shared memory (see XBD [Section 3.346](#), on page 89). It is
 62486 unspecified whether this function interoperates with the realtime interprocess communication
 62487 facilities defined in [Section 2.8](#) (on page 503).

62488 The *shmat()* function attaches the shared memory segment associated with the shared memory
 62489 identifier specified by *shmid* to the address space of the calling process. The segment is attached
 62490 at the address specified by one of the following criteria:

62491 If *shmaddr* is a null pointer, the segment is attached at the first available address as selected
 62492 by the system.

62493 If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is non-zero, the segment is
 62494 attached at the address given by (*shmaddr* - ((*uintptr_t*)*shmaddr* %SHMLBA)). The character
 62495 '%' is the C-language remainder operator.

62496 If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is 0, the segment is attached at
 62497 the address given by *shmaddr*.

62498 The segment is attached for reading if (*shmflg* &SHM_RDONLY) is non-zero and the
 62499 calling process has read permission; otherwise, if it is 0 and the calling process has read
 62500 and write permission, the segment is attached for reading and writing.

62501 **RETURN VALUE**

62502 Upon successful completion, *shmat()* shall increment the value of *shm_nattch* in the data
 62503 structure associated with the shared memory ID of the attached shared memory segment and
 62504 return the segment's start address. Also, the *shm_atime* timestamp shall be set to the current
 62505 time, as described in [Section 2.7.1](#) (on page 502).

62506 Otherwise, the shared memory segment shall not be attached, *shmat()* shall return (**void ***)-1,
 62507 and *errno* shall be set to indicate the error.

62508 **ERRORS**

62509 The *shmat()* function shall fail if:

62510 [EACCES] Operation permission is denied to the calling process; see [Section 2.7](#) (on page
 62511 501).

62512 [EINVAL] The value of *shmid* is not a valid shared memory identifier, the *shmaddr* is not a
 62513 null pointer, and the value of (*shmaddr* - ((*uintptr_t*)*shmaddr* %SHMLBA)) is an
 62514 illegal address for attaching shared memory; or the *shmaddr* is not a null
 62515 pointer, (*shmflg* &SHM_RND) is 0, and the value of *shmaddr* is an illegal
 62516 address for attaching shared memory.

62517 [EMFILE] The number of shared memory segments attached to the calling process
 62518 would exceed the system-imposed limit.

62519 [ENOMEM] The available data space is not large enough to accommodate the shared
 62520 memory segment.

62521 EXAMPLES

62522 None.

62523 APPLICATION USAGE

62524 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
62525 Application developers who need to use IPC should design their applications so that modules
62526 using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the
62527 alternative interfaces.

62528 RATIONALE

62529 None.

62530 FUTURE DIRECTIONS

62531 None.

62532 SEE ALSO

62533 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*,
62534 *shmget()*, *shm_open()*, *shm_unlink()*

62535 XBD [Section 3.346](#) (on page 89), [<sys/shm.h>](#)

62536 CHANGE HISTORY

62537 First released in Issue 2. Derived from Issue 2 of the SVID.

62538 Issue 5

62539 Moved from SHARED MEMORY to BASE.

62540 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
62541 DIRECTIONS to a new APPLICATION USAGE section.

62542 Issue 6

62543 The Open Group Corrigendum U021/13 is applied.

62544 Issue 7

62545 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0572 [345] is applied.

62546 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0327 [522] is applied.

62547 **NAME**

62548 shmctl — XSI shared memory control operations

62549 **SYNOPSIS**

```
62550 XSI #include <sys/shm.h>
62551 int shmctl(int shmid, int cmd, struct shm_id_ds *buf);
```

62552 **DESCRIPTION**

62553 The *shmctl()* function operates on XSI shared memory (see XBD [Section 3.346](#), on page 89). It is
 62554 unspecified whether this function interoperates with the realtime interprocess communication
 62555 facilities defined in [Section 2.8](#) (on page 503).

62556 The *shmctl()* function provides a variety of shared memory control operations as specified by
 62557 *cmd*. The following values for *cmd* are available:

62558 **IPC_STAT** Place the current value of each member of the **shm_id_ds** data structure
 62559 associated with *shmid* into the structure pointed to by *buf*. The contents of the
 62560 structure are defined in **<sys/shm.h>**.

62561 **IPC_SET** Set the value of the following members of the **shm_id_ds** data structure
 62562 associated with *shmid* to the corresponding value found in the structure
 62563 pointed to by *buf*:

```
62564 shm_perm.uid
62565 shm_perm.gid
62566 shm_perm.mode Low-order nine bits.
```

62567 Also, the *shm_ctime* timestamp shall be set to the current time, as described in
 62568 [Section 2.7.1](#) (on page 502).

62569 IPC_SET can only be executed by a process that has an effective user ID equal
 62570 to either that of a process with appropriate privileges or to the value of
 62571 *shm_perm.cuid* or *shm_perm.uid* in the **shm_id_ds** data structure associated with
 62572 *shmid*.

62573 **IPC_RMID** Remove the shared memory identifier specified by *shmid* from the system and
 62574 destroy the shared memory segment and **shm_id_ds** data structure associated
 62575 with it. IPC_RMID can only be executed by a process that has an effective user
 62576 ID equal to either that of a process with appropriate privileges or to the value
 62577 of *shm_perm.cuid* or *shm_perm.uid* in the **shm_id_ds** data structure associated
 62578 with *shmid*.

62579 **RETURN VALUE**

62580 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to
 62581 indicate the error.

62582 **ERRORS**

62583 The *shmctl()* function shall fail if:

62584 [EACCES] The argument *cmd* is equal to IPC_STAT and the calling process does not have
 62585 read permission; see [Section 2.7](#) (on page 501).

62586 [EINVAL] The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*
 62587 is not a valid command.

62588 [EPERM] The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID
 62589 of the calling process is not equal to that of a process with appropriate
 62590 privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in

62591 the data structure associated with *shmid*.
62592 The *shmctl()* function may fail if:
62593 [EOVERFLOW] The *cmd* argument is `IPC_STAT` and the *gid* or *uid* value is too large to be
62594 stored in the structure pointed to by the *buf* argument.

62595 EXAMPLES

62596 None.

62597 APPLICATION USAGE

62598 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
62599 Application developers who need to use IPC should design their applications so that modules
62600 using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the
62601 alternative interfaces.

62602 RATIONALE

62603 None.

62604 FUTURE DIRECTIONS

62605 None.

62606 SEE ALSO

62607 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), [shmact\(\)](#), [shmdt\(\)](#), [shmget\(\)](#), [shm_open\(\)](#),
62608 [shm_unlink\(\)](#)

62609 XBD [Section 3.346](#) (on page 89), [<sys/shm.h>](#)

62610 CHANGE HISTORY

62611 First released in Issue 2. Derived from Issue 2 of the SVID.

62612 Issue 5

62613 Moved from SHARED MEMORY to BASE.

62614 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
62615 DIRECTIONS to a new APPLICATION USAGE section.

62616 Issue 7

62617 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0573 [345] is applied.

62618 **NAME**

62619 shmdt — XSI shared memory detach operation

62620 **SYNOPSIS**

```
62621 XSI #include <sys/shm.h>
62622 int shmdt(const void *shmaddr);
```

62623 **DESCRIPTION**

62624 The *shmdt()* function operates on XSI shared memory (see XBD [Section 3.346](#), on page 89). It is
 62625 unspecified whether this function interoperates with the realtime interprocess communication
 62626 facilities defined in [Section 2.8](#) (on page 503).

62627 The *shmdt()* function detaches the shared memory segment located at the address specified by
 62628 *shmaddr* from the address space of the calling process.

62629 **RETURN VALUE**

62630 Upon successful completion, *shmdt()* shall decrement the value of *shm_nattch* in the data
 62631 structure associated with the shared memory ID of the attached shared memory segment and
 62632 return 0. Also, the *shm_dtime* timestamp shall be set to the current time, as described in [Section](#)
 62633 [2.7.1](#) (on page 502).

62634 Otherwise, the shared memory segment shall not be detached, *shmdt()* shall return -1 , and *errno*
 62635 shall be set to indicate the error.

62636 **ERRORS**

62637 The *shmdt()* function shall fail if:

62638	[EINVAL]	The value of <i>shmaddr</i> is not the data segment start address of a shared memory segment.
62639		

62640 **EXAMPLES**

62641 None.

62642 **APPLICATION USAGE**

62643 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 62644 Application developers who need to use IPC should design their applications so that modules
 62645 using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the
 62646 alternative interfaces.

62647 **RATIONALE**

62648 None.

62649 **FUTURE DIRECTIONS**

62650 None.

62651 **SEE ALSO**

62652 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), *exec*, *exit()*, *fork()*, *shmat()*, *shmctl()*,
 62653 *shmget()*, *shm_open()*, *shm_unlink()*

62654 XBD [Section 3.346](#) (on page 89), [<sys/shm.h>](#)

62655 **CHANGE HISTORY**

62656 First released in Issue 2. Derived from Issue 2 of the SVID.

62657 **Issue 5**

62658 Moved from SHARED MEMORY to BASE.

62659 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
62660 DIRECTIONS to a new APPLICATION USAGE section.

62661 **Issue 7**

62662 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0574 [345] is applied.

62663 **NAME**

62664 shmget — get an XSI shared memory segment

62665 **SYNOPSIS**

```
62666 XSI #include <sys/shm.h>
62667 int shmget(key_t key, size_t size, int shmflg);
```

62668 **DESCRIPTION**

62669 The *shmget()* function operates on XSI shared memory (see XBD [Section 3.346](#), on page 89). It is
 62670 unspecified whether this function interoperates with the realtime interprocess communication
 62671 facilities defined in [Section 2.8](#) (on page 503).

62672 The *shmget()* function shall return the shared memory identifier associated with *key*.

62673 A shared memory identifier, associated data structure, and shared memory segment of at least
 62674 *size* bytes (see [<sys/shm.h>](#)) are created for *key* if one of the following is true:

62675 The argument *key* is equal to `IPC_PRIVATE`.

62676 The argument *key* does not already have a shared memory identifier associated with it and
 62677 (*shmflg* & `IPC_CREAT`) is non-zero.

62678 Upon creation, the data structure associated with the new shared memory identifier shall be
 62679 initialized as follows:

62680 The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set to the
 62681 effective user ID and effective group ID, respectively, of the calling process.

62682 The low-order nine bits of *shm_perm.mode* are set to the low-order nine bits of *shmflg*.

62683 The value of *shm_segsz* is set to the value of *size*.

62684 The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set to 0.

62685 The value of *shm_ctime* is set to the current time, as described in [Section 2.7.1](#) (on page 502).

62686 When the shared memory segment is created, it shall be initialized with all zero values.

62687 **RETURN VALUE**

62688 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared
 62689 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

62690 **ERRORS**

62691 The *shmget()* function shall fail if:

62692 [EACCES] A shared memory identifier exists for *key* but operation permission as
 62693 specified by the low-order nine bits of *shmflg* would not be granted; see
 62694 [Section 2.7](#) (on page 501).

62695 [EEXIST] A shared memory identifier exists for the argument *key* but (*shmflg*
 62696 & `IPC_CREAT`) && (*shmflg* & `IPC_EXCL`) is non-zero.

62697 [EINVAL] A shared memory segment is to be created and the value of *size* is less than
 62698 the system-imposed minimum or greater than the system-imposed maximum.

62699 [EINVAL] No shared memory segment is to be created and a shared memory segment
 62700 exists for *key* but the size of the segment associated with it is less than *size*.

62701 [ENOENT] A shared memory identifier does not exist for the argument *key* and (*shmflg*
 62702 & `IPC_CREAT`) is 0.

62703 [ENOMEM] A shared memory identifier and associated shared memory segment are to be
62704 created, but the amount of available physical memory is not sufficient to fill
62705 the request.

62706 [ENOSPC] A shared memory identifier is to be created, but the system-imposed limit on
62707 the maximum number of allowed shared memory identifiers system-wide
62708 would be exceeded.

62709 EXAMPLES

62710 None.

62711 APPLICATION USAGE

62712 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
62713 Application developers who need to use IPC should design their applications so that modules
62714 using the IPC routines described in [Section 2.7](#) (on page 501) can be easily modified to use the
62715 alternative interfaces.

62716 RATIONALE

62717 None.

62718 FUTURE DIRECTIONS

62719 None.

62720 SEE ALSO

62721 [Section 2.7](#) (on page 501), [Section 2.8](#) (on page 503), [ftok\(\)](#), [shmat\(\)](#), [shmctl\(\)](#), [shmdt\(\)](#), [shm_open\(\)](#),
62722 [shm_unlink\(\)](#)

62723 XBD [Section 3.346](#) (on page 89), [<sys/shm.h>](#)

62724 CHANGE HISTORY

62725 First released in Issue 2. Derived from Issue 2 of the SVID.

62726 Issue 5

62727 Moved from SHARED MEMORY to BASE.

62728 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
62729 DIRECTIONS to a new APPLICATION USAGE section.

62730 Issue 7

62731 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0575 [345], XSH/TC1-2008/0576 [363],
62732 and XSH/TC1-2008/0577 [344] are applied.

62733 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0328 [640] is applied.

62734 **NAME**

62735 shutdown — shut down socket send and receive operations

62736 **SYNOPSIS**

62737 #include <sys/socket.h>
62738 int shutdown(int *socket*, int *how*);

62739 **DESCRIPTION**

62740 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket
62741 associated with the file descriptor *socket* to be shut down.

62742 The *shutdown()* function takes the following arguments:

- 62743 *socket* Specifies the file descriptor of the socket.
- 62744 *how* Specifies the type of shutdown. The values are as follows:
- 62745 SHUT_RD Disables further receive operations.
 - 62746 SHUT_WR Disables further send operations.
 - 62747 SHUT_RDWR Disables further send and receive operations.

62748 The *shutdown()* function disables subsequent send and/or receive operations on a socket,
62749 depending on the value of the *how* argument.

62750 **RETURN VALUE**

62751 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*
62752 set to indicate the error.

62753 **ERRORS**

- 62754 The *shutdown()* function shall fail if:
- 62755 [EBADF] The *socket* argument is not a valid file descriptor.
 - 62756 [EINVAL] The *how* argument is invalid.
 - 62757 [ENOTCONN] The socket is not connected.
 - 62758 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 62759 The *shutdown()* function may fail if:
- 62760 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

62761 **EXAMPLES**

62762 None.

62763 **APPLICATION USAGE**

62764 None.

62765 **RATIONALE**

62766 None.

62767 **FUTURE DIRECTIONS**

62768 None.

62769 **SEE ALSO**

62770 [getsockopt\(\)](#), [pselect\(\)](#), [read\(\)](#), [recv\(\)](#), [recvfrom\(\)](#), [recvmsg\(\)](#), [send\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#), [socket\(\)](#),
62771 [write\(\)](#)

62772 XBD [<sys/socket.h>](#)

62773 **CHANGE HISTORY**

62774 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

62775 **NAME**

62776 sigaction ‡examine and change a signal action

62777 **SYNOPSIS**

```
62778 CX #include <signal.h>
62779 int sigaction(int sig, const struct sigaction *restrict act,
62780 struct sigaction *restrict oact);
```

62781 **DESCRIPTION**

62782 The *sigaction()* function allows the calling process to examine and/or specify the action to be
 62783 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are
 62784 defined in **<signal.h>**.

62785 The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>**
 62786 header to include at least the following members:

Member Type	Member Name	Description
void(*) (int)	<i>sa_handler</i>	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
sigset_t	<i>sa_mask</i>	Additional set of signals to be blocked during execution of signal-catching function.
int	<i>sa_flags</i>	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	<i>sa_sigaction</i>	Pointer to a signal-catching function.

62796 The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
 62797 shall not use both simultaneously.

62798 If the argument *act* is not a null pointer, it points to a structure specifying the action to be
 62799 associated with the specified signal. If the argument *oact* is not a null pointer, the action
 62800 previously associated with the signal is stored in the location pointed to by the argument *oact*. If
 62801 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to
 62802 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall
 62803 not be added to the signal mask using this mechanism; this restriction shall be enforced by the
 62804 system without causing an error to be indicated.

62805 If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the
 62806 *sa_handler* field identifies the action to be associated with the specified signal. If the
 62807 SA_SIGINFO flag is set in the *sa_flags* field, the *sa_sigaction* field specifies a signal-catching
 62808 function.

62809 The *sa_flags* field can be used to modify the behavior of the specified signal.

62810 The following flags, defined in the **<signal.h>** header, can be set in *sa_flags*:

62811 XSI SA_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children
 62812 continue.

62813 If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is not set in *sa_flags*, and
 62814 the implementation supports the SIGCHLD signal, then a SIGCHLD
 62815 signal shall be generated for the calling process whenever any of its child
 62816 XSI processes stop and a SIGCHLD signal may be generated for the calling
 62817 process whenever any of its stopped child processes are continued. If *sig*
 62818 is SIGCHLD and the SA_NOCLDSTOP flag is set in *sa_flags*, then the
 62819 implementation shall not generate a SIGCHLD signal in this way.

62820	XSI	SA_ONSTACK	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
62821			
62822			
62823		SA_RESETHAND	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
62824			
62825		Note:	SIGILL and SIGTRAP cannot be automatically reset when delivered;
62826			the system silently enforces this restriction.
62827			Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
62828			
62829			In addition, if this flag is set, <i>sigaction()</i> may behave as if the SA_NODEFER flag were also set.
62830			
62831		SA_RESTART	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If an interruptible function which uses a timeout is restarted, the duration of the timeout following the restart is set to an unspecified value that does not exceed the original timeout value. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
62832			
62833			
62834			
62835			
62836			
62837			
62838			
62839		SA_SIGINFO	If cleared and the signal is caught, the signal-catching function shall be entered as:
62840			
62841			<pre>void func(int signo);</pre>
62842			where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.
62843			
62844			
62845			
62846			If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
62847			
62848			<pre>void func(int signo, siginfo_t *info, void *context);</pre>
62849			where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type siginfo_t explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type ucontext_t to refer to the receiving thread's context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member.
62850			
62851			
62852			
62853			
62854			
62855			
62856			
62857			The <i>si_signo</i> member contains the system-generated signal number.
62858	XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
62859			
62860			
62861			The <i>si_code</i> member contains a code identifying the cause of the signal, as described in Section 2.4.3 (on page 490).
62862			

62863 XSI SA_NOCLDWAIT If *sig* does not equal SIGCHLD, the behavior is unspecified. Otherwise,
 62864 the behavior of the SA_NOCLDWAIT flag is as specified in [Consequences](#)
 62865 [of Process Termination](#) (on page 553).

62866 SA_NODEFER If set and *sig* is caught, *sig* shall not be added to the thread's signal mask
 62867 on entry to the signal handler unless it is included in *sa_mask*. Otherwise,
 62868 *sig* shall always be added to the thread's signal mask on entry to the
 62869 signal handler.

62870 When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask
 62871 is calculated and installed for the duration of the signal-catching function (or until a call to either
 62872 *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current
 62873 signal mask and the value of the *sa_mask* for the signal being delivered, and unless
 62874 SA_NODEFER or SA_RESETHAND is set, then including the signal being delivered. If and
 62875 when the user's signal handler returns normally, the original signal mask is restored.

62876 Once an action is installed for a specific signal, it shall remain installed until another action is
 62877 explicitly requested (by another call to *sigaction()*), until the SA_RESETHAND flag causes
 62878 resetting of the handler, or until one of the *exec* functions is called.

62879 If the previous action for *sig* had been established by *signal()*, the values of the fields returned in
 62880 the structure pointed to by *oact* are unspecified, and in particular *oact->sa_handler* is not
 62881 necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a
 62882 copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the
 62883 signal shall be as if the original call to *signal()* were repeated.

62884 If *sigaction()* fails, no new signal handler is installed.

62885 It is unspecified whether an attempt to set the action for a signal that cannot be caught or
 62886 ignored to SIG_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

62887 If SA_SIGINFO is not set in *sa_flags*, then the disposition of subsequent occurrences of *sig* when
 62888 it is already pending is implementation-defined; the signal-catching function shall be invoked
 62889 with a single argument. If SA_SIGINFO is set in *sa_flags*, then subsequent occurrences of *sig*
 62890 generated by *sigqueue()* or as a result of any signal-generating function that supports the
 62891 specification of an application-defined value (when *sig* is already pending) shall be queued in
 62892 FIFO order until delivered or accepted; the signal-catching function shall be invoked with three
 62893 arguments. The application specified value is passed to the signal-catching function as the
 62894 *si_value* member of the **siginfo_t** structure.

62895 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the
 62896 same signal is unspecified.

62897 **RETURN VALUE**

62898 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall
 62899 be set to indicate the error, and no new signal-catching function shall be installed.

62900 **ERRORS**62901 The *sigaction()* function shall fail if:62902 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
62903 signal that cannot be caught or ignore a signal that cannot be ignored.62904 The *sigaction()* function may fail if:62905 [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
62906 caught or ignored (or both).62907 In addition, on systems that do not support the XSI option, the *sigaction()* function may fail if the
62908 SA_SIGINFO flag is set in the *sa_flags* field of the **sigaction** structure for a signal not in the range
62909 SIGRTMIN to SIGRTMAX.62910 **EXAMPLES**62911 **Establishing a Signal Handler**62912 The following example demonstrates the use of *sigaction()* to establish a handler for the SIGINT
62913 signal.

```

62914 #include <signal.h>
62915 static void handler(int signum)
62916 {
62917     /* Take appropriate actions for signal delivery */
62918 }
62919 int main()
62920 {
62921     struct sigaction sa;
62922     sa.sa_handler = handler;
62923     sigemptyset(&sa.sa_mask);
62924     sa.sa_flags = SA_RESTART; /* Restart functions if
62925                             interrupted by handler */
62926     if (sigaction(SIGINT, &sa, NULL) == -1)
62927         /* Handle error */;
62928     /* Further code */
62929 }

```

62930 **APPLICATION USAGE**

62931 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In
62932 particular, *sigaction()* and *signal()* should not be used in the same process to control the same
62933 signal. The behavior of async-signal-safe functions, as defined in their respective
62934 DESCRIPTION sections, is as specified by this volume of POSIX.1-2017, regardless of invocation
62935 from a signal-catching function. This is the only intended meaning of the statement that async-
62936 signal-safe functions may be used in signal-catching functions without restrictions. Applications
62937 must still consider all effects of such functions on such things as data structures, files, and
62938 process state. In particular, application developers need to consider the restrictions on
62939 interactions when interrupting *sleep()* and interactions among multiple handles for a file
62940 description. The fact that any specific function is listed as async-signal-safe does not necessarily
62941 mean that invocation of that function from a signal-catching function is recommended.

62942 In order to prevent errors arising from interrupting non-async-signal-safe function calls,
62943 applications should protect calls to these functions either by blocking the appropriate signals or

62944 through the use of some programmatic semaphore (see [semget\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#), and so
 62945 on). Note in particular that even the “safe” functions may modify *errno*; the signal-catching
 62946 function, if not executing as an independent thread, should save and restore its value in order to
 62947 avoid the possibility that delivery of a signal in between an error return from a function that sets
 62948 *errno* and the subsequent examination of *errno* could result in the signal-catching function
 62949 changing the value of *errno*. Naturally, the same principles apply to the async-signal-safety of
 62950 application routines and asynchronous data access. Note that [longjmp\(\)](#) and [siglongjmp\(\)](#) are not
 62951 in the list of async-signal-safe functions. This is because the code executing after [longjmp\(\)](#) and
 62952 [siglongjmp\(\)](#) can call any unsafe functions with the same danger as calling those unsafe
 62953 functions directly from the signal handler. Applications that use [longjmp\(\)](#) and [siglongjmp\(\)](#) from
 62954 within signal handlers require rigorous protection in order to be portable. Many of the other
 62955 functions that are excluded from the list are traditionally implemented using either [malloc\(\)](#) or
 62956 [free\(\)](#) functions or the standard I/O library, both of which traditionally use data structures in a
 62957 non-async-signal-safe manner. Since any combination of different functions using a common
 62958 data structure can cause async-signal-safety problems, this volume of POSIX.1-2017 does not
 62959 define the behavior when any unsafe function is called in a signal handler that interrupts an
 62960 unsafe function.

62961 Usually, the signal is executed on the stack that was in effect before the signal was delivered. An
 62962 alternate stack may be specified to receive a subset of the signals being caught.

62963 When the signal handler returns, the receiving thread resumes execution at the point it was
 62964 interrupted unless the signal handler makes other arrangements. If [longjmp\(\)](#) or [_longjmp\(\)](#) is
 62965 used to leave the signal handler, then the signal mask must be explicitly restored.

62966 This volume of POSIX.1-2017 defines the third argument of a signal handling function when
 62967 SA_SIGINFO is set as a **void *** instead of a **ucontext_t ***, but without requiring type checking.
 62968 New applications should explicitly cast the third argument of the signal handling function to
 62969 **ucontext_t ***.

62970 The BSD optional four argument signal handling function is not supported by this volume of
 62971 POSIX.1-2017. The BSD declaration would be:

```
62972 void handler(int sig, int code, struct sigcontext *scp,  
62973             char *addr);
```

62974 where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer
 62975 to the **sigcontext** structure, and *addr* is additional address information. Much the same
 62976 information is available in the objects pointed to by the second argument of the signal handler
 62977 specified when SA_SIGINFO is set.

62978 Since the [sigaction\(\)](#) function is allowed but not required to set SA_NODEFER when the
 62979 application sets the SA_RESETHAND flag, applications which depend on the SA_RESETHAND
 62980 functionality for the newly installed signal handler must always explicitly set SA_NODEFER
 62981 when they set SA_RESETHAND in order to be portable.

62982 See also the rationale for Realtime Signal Generation and Delivery in XRAT [Section B.2.4.2](#) (on
 62983 page 3578).

62984 RATIONALE

62985 Although this volume of POSIX.1-2017 requires that signals that cannot be ignored shall not be
 62986 added to the signal mask when a signal-catching function is entered, there is no explicit
 62987 requirement that subsequent calls to [sigaction\(\)](#) reflect this in the information returned in the *oact*
 62988 argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified
 62989 whether or not a subsequent call to [sigaction\(\)](#) returns with SIGKILL included in the *sa_mask*
 62990 field of *oact*.

62991 The SA_NOCLDSTOP flag, when supplied in the *act->sa_flags* parameter, allows overloading
 62992 SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated
 62993 child. Most conforming applications that catch SIGCHLD are expected to install signal-catching
 62994 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on
 62995 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of
 62996 interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the
 62997 signal-catching routine when they stop.

62998 Some historical implementations also define other mechanisms for stopping processes, such as
 62999 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when
 63000 processes stop due to this mechanism; however, that is beyond the scope of this volume of
 63001 POSIX.1-2017.

63002 This volume of POSIX.1-2017 requires that calls to *sigaction()* that supply a NULL *act* argument
 63003 succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or
 63004 SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases
 63005 and, in this respect, their behavior varies from *sigaction()*.

63006 This volume of POSIX.1-2017 requires that *sigaction()* properly save and restore a signal action
 63007 set up by the ISO C standard *signal()* function. However, there is no guarantee that the reverse is
 63008 true, nor could there be given the greater amount of information conveyed by the **sigaction**
 63009 structure. Because of this, applications should avoid using both functions for the same signal in
 63010 the same process. Since this cannot always be avoided in case of general-purpose library
 63011 routines, they should always be implemented with *sigaction()*.

63012 It was intended that the *signal()* function should be implementable as a library routine using
 63013 *sigaction()*.

63014 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990
 63015 standard to allow the application to request on a per-signal basis via an additional signal action
 63016 flag that the extra parameters, including the application-defined signal value, if any, be passed to
 63017 the signal-catching function.

63018 FUTURE DIRECTIONS

63019 None.

63020 SEE ALSO

63021 [Section 2.4](#) (on page 488), *exec*, *_Exit()*, *kill()*, *_longjmp()*, *longjmp()*, *pthread_sigmask()*, *raise()*,
 63022 *semget()*, *sem_init()*, *sem_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,
 63023 *sigismember()*, *signal()*, *sigsuspend()*, *wait()*, *waitid()*

63024 XBD [<signal.h>](#)

63025 CHANGE HISTORY

63026 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63027 Issue 5

63028 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX
 63029 Threads Extension.

63030 In the DESCRIPTION, the second argument to *func* when SA_SIGINFO is set is no longer
 63031 permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by
 63032 reference to [<signal.h>](#).

63033 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]
 63034 error is deleted.

63035 **Issue 6**

63036 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on
63037 Other Functions”, a reference to *sigpending()* is added.

63038 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal
63039 Effects on Other Functions” are moved to a separate section of this volume of POSIX.1-2017.

63040 Text describing functionality from the Realtime Signals Extension option is marked.

63041 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

63042 The [ENOTSUP] error condition is added.

63043 The normative text is updated to avoid use of the term “must” for application requirements.

63044 The **restrict** keyword is added to the *sigaction()* prototype for alignment with the
63045 ISO/IEC 9899: 1999 standard.

63046 References to the *wait3()* function are removed.

63047 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an
63048 extension over the ISO C standard.

63049 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table
63050 describing the **sigaction** structure.

63051 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/127 is applied, removing text from the
63052 DESCRIPTION duplicated later in the same section.

63053 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/128 is applied, updating the
63054 DESCRIPTION and APPLICATION USAGE sections. Changes are made to refer to the thread
63055 rather than the process.

63056 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/129 is applied, adding the example to the
63057 EXAMPLES section.

63058 **Issue 7**

63059 Austin Group Interpretation 1003.1-2001 #004 is applied.

63060 Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the
63061 SA_NODEFER flag with respect to the signal mask, and clarifying the SA_RESTART flag for
63062 interrupted functions which use timeouts.

63063 Austin Group Interpretation 1003.1-2001 #156 is applied.

63064 SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section.

63065 SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement
63066 that when the SA_RESETHAND flag is set, *sigaction()* shall behave as if the SA_NODEFER flag
63067 were also set.

63068 Functionality relating to the Realtime Signals Extension option is moved to the Base.

63069 The description of the *si_code* member is replaced with a reference to [Section 2.4.3](#) (on page 490).

63070 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0578 [66] and XSH/TC1-2008/0579
63071 [140] are applied.

63072 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0329 [690] and XSH/TC2-2008/0330
63073 [491] are applied.

63074 **NAME**

63075 sigaddset ‡add a signal to a signal set

63076 **SYNOPSIS**

```
63077 CX #include <signal.h>
63078 int sigaddset(sigset_t *set, int signo);
```

63079 **DESCRIPTION**

63080 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed
63081 to by *set*.

63082 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type
63083 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
63084 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
63085 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
63086 *sigwaitinfo()*, the results are undefined.

63087 **RETURN VALUE**

63088 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*
63089 to indicate the error.

63090 **ERRORS**

63091 The *sigaddset()* function may fail if:

63092 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

63093 **EXAMPLES**

63094 None.

63095 **APPLICATION USAGE**

63096 None.

63097 **RATIONALE**

63098 None.

63099 **FUTURE DIRECTIONS**

63100 None.

63101 **SEE ALSO**

63102 Section 2.4 (on page 488), *pthread_sigmask()*, *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,
63103 *sigismember()*, *sigpending()*, *sigsuspend()*

63104 XBD <signal.h>

63105 **CHANGE HISTORY**

63106 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63107 **Issue 5**

63108 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
63109 previous issues.

63110 **Issue 6**

63111 The normative text is updated to avoid use of the term “must” for application requirements.

63112 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an
63113 extension over the ISO C standard.

63114 **NAME**

63115 sigaltstack ‡'set and get signal alternate stack context

63116 **SYNOPSIS**

```
63117 XSI #include <signal.h>
63118 int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

63119 **DESCRIPTION**

63120 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack
 63121 for signal handlers for the current thread. Signals that have been explicitly declared to execute
 63122 on the alternate stack shall be delivered on the alternate stack.

63123 If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack
 63124 that shall take effect upon return from *sigaltstack()*. The *ss_flags* member specifies the new stack
 63125 state. If it is set to *SS_DISABLE*, the stack is disabled and *ss_sp* and *ss_size* are ignored.
 63126 Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address
 63127 and size of the stack.

63128 The range of addresses starting at *ss_sp* up to but not including *ss_sp+ss_size* is available to the
 63129 implementation for use as the stack. This function makes no assumptions regarding which end
 63130 is the stack base and in which direction the stack grows as items are pushed.

63131 If *oss* is not a null pointer, upon successful completion it shall point to a **stack_t** structure that
 63132 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss_sp*
 63133 and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the
 63134 stack's state, and may contain one of the following values:

63135 **SS_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to
 63136 modify the alternate signal stack while the process is executing on it fail. This
 63137 flag shall not be modified by processes.

63138 **SS_DISABLE** The alternate signal stack is currently disabled.

63139 The value *SIGSTKSZ* is a system default specifying the number of bytes that would be used to
 63140 cover the usual case when manually allocating an alternate stack area. The value *MINSIGSTKSZ*
 63141 is defined to be the minimum stack size for a signal handler. In computing an alternate stack
 63142 size, a program should add that amount to its stack requirements to allow for the system
 63143 implementation overhead. The constants *SS_ONSTACK*, *SS_DISABLE*, *SIGSTKSZ*, and
 63144 *MINSIGSTKSZ* are defined in **<signal.h>**.

63145 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new
 63146 process image.

63147 In some implementations, a signal (whether or not indicated to execute on the alternate stack)
 63148 shall always execute on the alternate stack if it is delivered while another signal is being caught
 63149 using the alternate stack.

63150 Use of this function by library threads that are not bound to kernel-scheduled entities results in
 63151 undefined behavior.

63152 **RETURN VALUE**

63153 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return -1 and set *errno*
 63154 to indicate the error.

63155 **ERRORS**63156 The *sigaltstack()* function shall fail if:63157 [EINVAL] The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss*
63158 contains flags other than *SS_DISABLE*.63159 [ENOMEM] The size of the alternate stack area is less than *MINSIGSTKSZ*.

63160 [EPERM] An attempt was made to modify an active stack.

63161 **EXAMPLES**63162 **Allocating Memory for an Alternate Stack**

63163 The following example illustrates a method for allocating memory for an alternate stack.

```

63164 #include <signal.h>
63165 ...
63166 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
63167     /* Error return. */
63168     sigstk.ss_size = SIGSTKSZ;
63169     sigstk.ss_flags = 0;
63170     if (sigaltstack(&sigstk, (stack_t *)0) < 0)
63171         perror("sigaltstack");

```

63172 **APPLICATION USAGE**

63173 On some implementations, stack space is automatically extended as needed. On those
63174 implementations, automatic extension is typically not available for an alternate stack. If the stack
63175 overflows, the behavior is undefined.

63176 **RATIONALE**

63177 None.

63178 **FUTURE DIRECTIONS**

63179 None.

63180 **SEE ALSO**63181 [Section 2.4](#) (on page 488), *exec*, *sigaction()*, *sigsetjmp()*63182 XBD [<signal.h>](#)63183 **CHANGE HISTORY**

63184 First released in Issue 4, Version 2.

63185 **Issue 5**

63186 Moved from X/OPEN UNIX extension to BASE.

63187 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in
63188 previous issues.

63189 **Issue 6**

63190 The normative text is updated to avoid use of the term “must” for application requirements.

63191 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the
63192 ISO/IEC 9899:1999 standard.

63193 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence
63194 to include “for the current thread”.

63195 **NAME**

63196 sigdelset — delete a signal from a signal set

63197 **SYNOPSIS**

```
63198 CX #include <signal.h>
63199 int sigdelset(sigset_t *set, int signo);
```

63200 **DESCRIPTION**

63201 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set
 63202 pointed to by *set*.

63203 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
 63204 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
 63205 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
 63206 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
 63207 *sigwaitinfo()*, the results are undefined.

63208 **RETURN VALUE**

63209 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*
 63210 to indicate the error.

63211 **ERRORS**

63212 The *sigdelset()* function may fail if:

63213 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal
 63214 number.

63215 **EXAMPLES**

63216 None.

63217 **APPLICATION USAGE**

63218 None.

63219 **RATIONALE**

63220 None.

63221 **FUTURE DIRECTIONS**

63222 None.

63223 **SEE ALSO**

63224 Section 2.4 (on page 488), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*,
 63225 *sigismember()*, *sigpending()*, *sigsuspend()*

63226 XBD [<signal.h>](#)

63227 **CHANGE HISTORY**

63228 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63229 **Issue 5**

63230 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
 63231 previous issues.

63232 **Issue 6**

63233 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an
 63234 extension over the ISO C standard.

63235 **NAME**

63236 sigemptyset ‡initialize and empty a signal set

63237 **SYNOPSIS**

```
63238 CX #include <signal.h>
63239 int sigemptyset(sigset_t *set);
```

63240 **DESCRIPTION**

63241 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined
63242 in POSIX.1-2017 are excluded.

63243 **RETURN VALUE**

63244 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set
63245 *errno* to indicate the error.

63246 **ERRORS**

63247 No errors are defined.

63248 **EXAMPLES**

63249 None.

63250 **APPLICATION USAGE**

63251 None.

63252 **RATIONALE**

63253 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or set)
63254 all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the
63255 structure, such as a version field, to permit binary-compatibility between releases where the size
63256 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any
63257 other use of the signal set, even if such use is read-only (for example, as an argument to
63258 *sigpending()*). This function is not intended for dynamic allocation.

63259 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or
63260 exclude) all the signals defined in this volume of POSIX.1-2017. Although it is outside the scope
63261 of this volume of POSIX.1-2017 to place this requirement on signals that are implemented as
63262 extensions, it is recommended that implementation-defined signals also be affected by these
63263 functions. However, there may be a good reason for a particular signal not to be affected. For
63264 example, blocking or ignoring an implementation-defined signal may have undesirable side-
63265 effects, whereas the default action for that signal is harmless. In such a case, it would be
63266 preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.

63267 In early proposals there was no distinction between invalid and unsupported signals (the names
63268 of optional signals that were not supported by an implementation were not defined by that
63269 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.
63270 With that distinction, it is not necessary to require implementations of these functions to
63271 determine whether an optional signal is actually supported, as that could have a significant
63272 performance impact for little value. The error could have been required for invalid signals and
63273 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is
63274 optional in both cases.

63275 **FUTURE DIRECTIONS**

63276 None.

63277 **SEE ALSO**

63278 Section 2.4 (on page 488), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,
63279 *sigismember()*, *sigpending()*, *sigsuspend()*

63280 XBD [<signal.h>](#)

63281 **CHANGE HISTORY**

63282 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63283 **Issue 6**

63284 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an
63285 extension over the ISO C standard.

63286 **NAME**

63287 sigfillset ‡initialize and fill a signal set

63288 **SYNOPSIS**

```
63289 CX #include <signal.h>
63290 int sigfillset(sigset_t *set);
```

63291 **DESCRIPTION**

63292 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals
63293 defined in this volume of POSIX.1-2017 are included.

63294 **RETURN VALUE**

63295 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*
63296 to indicate the error.

63297 **ERRORS**

63298 No errors are defined.

63299 **EXAMPLES**

63300 None.

63301 **APPLICATION USAGE**

63302 None.

63303 **RATIONALE**

63304 Refer to *sigemptyset()* (on page 1961).

63305 **FUTURE DIRECTIONS**

63306 None.

63307 **SEE ALSO**

63308 Section 2.4 (on page 488), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*,
63309 *sigismember()*, *sigpending()*, *sigsuspend()*

63310 XBD [<signal.h>](#)

63311 **CHANGE HISTORY**

63312 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63313 **Issue 6**

63314 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an
63315 extension over the ISO C standard.

63316 **NAME**

63317 sighold, sigignore, sigpause, sigrelse, sigset — signal management

63318 **SYNOPSIS**

```

63319 OB XSI #include <signal.h>
63320 int sighold(int sig);
63321 int sigignore(int sig);
63322 int sigpause(int sig);
63323 int sigrelse(int sig);
63324 void (*sigset(int sig, void (*disp)(int)))(int);

```

63325 **DESCRIPTION**

63326 Use of any of these functions is unspecified in a multi-threaded process.

63327 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal
63328 management.

63329 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal,
63330 which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's
63331 disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If *sigset()* is
63332 used, and *disp* is the address of a signal handler, the system shall add *sig* to the signal mask of
63333 the calling process before executing the signal handler; when the signal handler returns, the
63334 system shall restore the signal mask of the calling process to its state prior to the delivery of the
63335 signal. In addition, if *sigset()* is used, and *disp* is equal to SIG_HOLD, *sig* shall be added to the
63336 signal mask of the calling process and *sig*'s disposition shall remain unchanged. If *sigset()* is
63337 used, and *disp* is not equal to SIG_HOLD, *sig* shall be removed from the signal mask of the
63338 calling process.

63339 The *sighold()* function shall add *sig* to the signal mask of the calling process.63340 The *sigrelse()* function shall remove *sig* from the signal mask of the calling process.63341 The *sigignore()* function shall set the disposition of *sig* to SIG_IGN.

63342 The *sigpause()* function shall remove *sig* from the signal mask of the calling process and suspend
63343 the calling process until a signal is received. The *sigpause()* function shall restore the signal mask
63344 of the process to its original state before returning.

63345 If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes
63346 shall not be transformed into zombie processes when they terminate. If the calling process
63347 subsequently waits for its children, and the process has no unwaited-for children that were
63348 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*,
63349 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

63350 **RETURN VALUE**

63351 Upon successful completion, *sigset()* shall return SIG_HOLD if the signal had been blocked and
63352 the signal's previous disposition if it had not been blocked. Otherwise, SIG_ERR shall be
63353 returned and *errno* set to indicate the error.

63354 The *sigpause()* function shall suspend execution of the thread until a signal is received,
63355 whereupon it shall return -1 and set *errno* to [EINTR].

63356 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be
63357 returned and *errno* set to indicate the error.

63358 **ERRORS**

63359 These functions shall fail if:

63360 [EINVAL] The *sig* argument is an illegal signal number.63361 The *sigset()* and *sigignore()* functions shall fail if:63362 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a
63363 signal that cannot be ignored.63364 **EXAMPLES**

63365 None.

63366 **APPLICATION USAGE**63367 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling
63368 signals; new applications should use the *sigaction()* function instead of the obsolescent *sigset()*
63369 function.63370 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish
63371 critical regions of code that require the delivery of a signal to be temporarily deferred. For
63372 broader portability, the *pthread_sigmask()* or *sigprocmask()* functions should be used instead of
63373 the obsolescent *sighold()* and *sigrelse()* functions.63374 For broader portability, the *sigsuspend()* function should be used instead of the obsolescent
63375 *sigpause()* function.63376 **RATIONALE**63377 Each of these historic functions has a direct analog in the other functions which are required to
63378 be per-thread and thread-safe (aside from *sigprocmask()*, which is replaced by *pthread_sigmask()*).
63379 The *sigset()* function can be implemented as a simple wrapper for *sigaction()*. The *sighold()*
63380 function is equivalent to *sigprocmask()* or *pthread_sigmask()* with SIG_BLOCK set. The *sigignore()*
63381 function is equivalent to *sigaction()* with SIG_IGN set. The *sigpause()* function is equivalent to
63382 *sigsuspend()*. The *sigrelse()* function is equivalent to *sigprocmask()* or *pthread_sigmask()* with
63383 SIG_UNBLOCK set.63384 **FUTURE DIRECTIONS**

63385 These functions may be removed in a future version.

63386 **SEE ALSO**63387 Section 2.4 (on page 488), *exec*, *pause()*, *pthread_sigmask()*, *sigaction()*, *signal()*, *sigsuspend()*,
63388 *wait()*, *waitid()*

63389 XBD <signal.h>

63390 **CHANGE HISTORY**

63391 First released in Issue 4, Version 2.

63392 **Issue 5**

63393 Moved from X/OPEN UNIX extension to BASE.

63394 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of
63395 the process to its original state before returning.63396 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends
63397 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to
63398 [EINTR].

63399 **Issue 6**

63400 The normative text is updated to avoid use of the term “must” for application requirements.

63401 References to the *wait3()* function are removed.

63402 The XSI functions are split out into their own reference page.

63403 **Issue 7**

63404 SD5-XSH-ERN-113 and SD5-XSH-ERN-42 are applied, marking these functions obsolescent and
63405 updating the APPLICATION USAGE and RATIONALE sections.

63406 **NAME**

63407 siginterrupt — allow signals to interrupt functions

63408 **SYNOPSIS**

```
63409 OB XSI #include <signal.h>
63410 int siginterrupt(int sig, int flag);
```

63411 **DESCRIPTION**

63412 The *siginterrupt()* function shall change the restart behavior when a function is interrupted by
 63413 the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```
63414 int siginterrupt(int sig, int flag) {
63415     int ret;
63416     struct sigaction act;
63417
63418     (void) sigaction(sig, NULL, &act);
63419     if (flag)
63420         act.sa_flags &= ~SA_RESTART;
63421     else
63422         act.sa_flags |= SA_RESTART;
63423     ret = sigaction(sig, &act, NULL);
63424     return ret;
63425 }
```

63425 **RETURN VALUE**

63426 Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and
 63427 *errno* set to indicate the error.

63428 **ERRORS**

63429 The *siginterrupt()* function shall fail if:

63430 [EINVAL] The *sig* argument is not a valid signal number.

63431 **EXAMPLES**

63432 None.

63433 **APPLICATION USAGE**

63434 The *siginterrupt()* function supports programs written to historical system interfaces.
 63435 Applications should use the *sigaction()* with the SA_RESTART flag instead of the obsolescent
 63436 *siginterrupt()* function.

63437 **RATIONALE**

63438 None.

63439 **FUTURE DIRECTIONS**

63440 None.

63441 **SEE ALSO**

63442 [Section 2.4](#) (on page 488), *sigaction()*

63443 XBD [<signal.h>](#)

63444 **CHANGE HISTORY**

63445 First released in Issue 4, Version 2.

63446 **Issue 5**

63447 Moved from X/OPEN UNIX extension to BASE.

63448 **Issue 6**

63449 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in
63450 the sample implementation given in the DESCRIPTION.

63451 **Issue 7**

63452 The *siginterrupt()* function is marked obsolescent.

63453 **NAME**

63454 sigismember ‡test for a signal in a signal set

63455 **SYNOPSIS**

```
63456 CX #include <signal.h>
63457 int sigismember(const sigset_t *set, int signo);
```

63458 **DESCRIPTION**

63459 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set
 63460 pointed to by *set*.

63461 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
 63462 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
 63463 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
 63464 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
 63465 *sigwaitinfo()*, the results are undefined.

63466 **RETURN VALUE**

63467 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of
 63468 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.

63469 **ERRORS**

63470 The *sigismember()* function may fail if:

63471 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal
 63472 number.

63473 **EXAMPLES**

63474 None.

63475 **APPLICATION USAGE**

63476 None.

63477 **RATIONALE**

63478 None.

63479 **FUTURE DIRECTIONS**

63480 None.

63481 **SEE ALSO**

63482 Section 2.4 (on page 488), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,
 63483 *sigemptyset()*, *sigpending()*, *sigsuspend()*

63484 XBD <signal.h>

63485 **CHANGE HISTORY**

63486 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63487 **Issue 5**

63488 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
 63489 previous issues.

63490 **Issue 6**

63491 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an
 63492 extension over the ISO C standard.

63493 **NAME**

63494 siglongjmp ‡non-local goto with signal handling

63495 **SYNOPSIS**

```
63496 CX #include <setjmp.h>
63497 void siglongjmp(sigjmp_buf env, int val);
```

63498 **DESCRIPTION**63499 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:63500 References to *setjmp()* shall be equivalent to *sigsetjmp()*.

63501 The *siglongjmp()* function shall restore the saved signal mask if and only if the *env*
 63502 argument was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

63503 **RETURN VALUE**

63504 After *siglongjmp()* is completed, program execution shall continue as if the corresponding
 63505 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function
 63506 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

63507 **ERRORS**

63508 No errors are defined.

63509 **EXAMPLES**

63510 None.

63511 **APPLICATION USAGE**

63512 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant
 63513 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

63514 **RATIONALE**

63515 None.

63516 **FUTURE DIRECTIONS**

63517 None.

63518 **SEE ALSO**63519 *longjmp()*, *pthread_sigmask()*, *setjmp()*, *sigsetjmp()*, *sigsuspend()*63520 XBD [<setjmp.h>](#)63521 **CHANGE HISTORY**

63522 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

63523 **Issue 5**

63524 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

63525 **Issue 6**63526 The DESCRIPTION is rewritten in terms of *longjmp()*.

63527 The SYNOPSIS is marked CX since the presence of this function in the [<setjmp.h>](#) header is an
 63528 extension over the ISO C standard.

63529 **NAME**63530 `signal` ‡signal management63531 **SYNOPSIS**63532 `#include <signal.h>`63533 `void (*signal(int sig, void (*func)(int)))(int);`63534 **DESCRIPTION**

63535 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 63536 conflict between the requirements described here and the ISO C standard is unintentional. This
 63537 volume of POSIX.1-2017 defers to the ISO C standard.

63538 The `signal()` function chooses one of three ways in which receipt of the signal number `sig` is to be
 63539 subsequently handled. If the value of `func` is `SIG_DFL`, default handling for that signal shall
 63540 occur. If the value of `func` is `SIG_IGN`, the signal shall be ignored. Otherwise, the application
 63541 shall ensure that `func` points to a function to be called when that signal occurs. An invocation of
 63542 such a function because of a signal, or (recursively) of any further functions called by that
 63543 invocation (other than functions in the standard library), is called a “signal handler”.

63544 When a signal occurs, and `func` points to a function, it is implementation-defined whether the
 63545 equivalent of a:

63546 `signal(sig, SIG_DFL);`

63547 is executed or the implementation prevents some implementation-defined set of signals (at least
 63548 including `sig`) from occurring until the current signal handling has completed. (If the value of `sig`
 63549 is `SIGILL`, the implementation may alternatively define that no action is taken.) Next the
 63550 equivalent of:

63551 `(*func)(sig);`

63552 is executed. If and when the function returns, if the value of `sig` was `SIGFPE`, `SIGILL`, or
 63553 `SIGSEGV` or any other implementation-defined value corresponding to a computational
 63554 exception, the behavior is undefined. Otherwise, the program shall resume execution at the
 63555 point it was interrupted. The ISO C standard places a restriction on applications relating to the
 63556 use of `raise()` from signal handlers. This restriction does not apply to POSIX applications, as
 63557 POSIX.1-2017 requires `raise()` to be async-signal-safe (see [Section 2.4.3](#), on page 490).

63558 CX If the process is multi-threaded, or if the process is single-threaded and a signal handler is
 63559 executed other than as the result of:

63560 CX The process calling `abort()`, `raise()`, `kill()`, `pthread_kill()`, or `sigqueue()` to generate a signal
 63561 that is not blocked

63562 CX A pending signal being unblocked and being delivered before the call that unblocked it
 63563 returns

63564 CX the behavior is undefined if the signal handler refers to any object other than `errno` with static
 63565 storage duration other than by assigning a value to an object declared as `volatile sig_atomic_t`,
 63566 or if the signal handler calls any function defined in this standard other than one of the
 63567 functions listed in [Section 2.4](#) (on page 488).

63568 At program start-up, the equivalent of:

63569 `signal(sig, SIG_IGN);`

63570 is executed for some signals, and the equivalent of:

63571 `signal(sig, SIG_DFL);`

- 63572 CX is executed for all other signals (see *exec*).
- 63573 The *signal()* function shall not change the setting of *errno* if successful.
- 63574 **RETURN VALUE**
- 63575 If the request can be honored, *signal()* shall return the value of *func* for the most recent call to
- 63576 *signal()* for the specified signal *sig*. Otherwise, SIG_ERR shall be returned and a positive value
- 63577 shall be stored in *errno*.
- 63578 **ERRORS**
- 63579 The *signal()* function shall fail if:
- 63580 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
- 63581 signal that cannot be caught or ignore a signal that cannot be ignored.
- 63582 The *signal()* function may fail if:
- 63583 CX [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
- 63584 caught or ignored (or both).
- 63585 **EXAMPLES**
- 63586 None.
- 63587 **APPLICATION USAGE**
- 63588 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling
- 63589 signals; new applications should use *sigaction()* rather than *signal()*.
- 63590 **RATIONALE**
- 63591 None.
- 63592 **FUTURE DIRECTIONS**
- 63593 None.
- 63594 **SEE ALSO**
- 63595 [Section 2.4](#) (on page 488), *exec*, *pause()*, *raise()*, *sigaction()*, *sigsuspend()*, *waitid()*
- 63596 XBD [<signal.h>](#)
- 63597 **CHANGE HISTORY**
- 63598 First released in Issue 1. Derived from Issue 1 of the SVID.
- 63599 **Issue 5**
- 63600 Moved from X/OPEN UNIX extension to BASE.
- 63601 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of
- 63602 the process to its original state before returning.
- 63603 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends
- 63604 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to
- 63605 [EINTR].
- 63606 **Issue 6**
- 63607 Extensions beyond the ISO C standard are marked.
- 63608 The normative text is updated to avoid use of the term “must” for application requirements.
- 63609 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.
- 63610 References to the *wait3()* function are removed.
- 63611 The *sighold()*, *sigignore()*, *sigrelse()*, and *sigset()* functions are split out onto their own reference
- 63612 page.

63613 **Issue 7**

63614 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0580 [275], XSH/TC1-2008/0581 [66],
63615 and XSH/TC1-2008/0582 [105] are applied.

63616 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0331 [785] is applied.

63617 **NAME**63618 signbit \ddagger 'test sign63619 **SYNOPSIS**

63620 #include <math.h>

63621 int signbit(real-floating x);

63622 **DESCRIPTION**

63623 CX The functionality described on this reference page is aligned with the ISO C standard. Any
63624 conflict between the requirements described here and the ISO C standard is unintentional. This
63625 volume of POSIX.1-2017 defers to the ISO C standard.

63626 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,
63627 zeros, and infinities have a sign bit.

63628 **RETURN VALUE**

63629 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is
63630 negative.

63631 **ERRORS**

63632 No errors are defined.

63633 **EXAMPLES**

63634 None.

63635 **APPLICATION USAGE**

63636 None.

63637 **RATIONALE**

63638 None.

63639 **FUTURE DIRECTIONS**

63640 None.

63641 **SEE ALSO**63642 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *isnormal()*

63643 XBD <math.h>

63644 **CHANGE HISTORY**

63645 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

63646 **NAME**

63647 signgam †'log gamma function

63648 **SYNOPSIS**

```
63649 XSI #include <math.h>  
63650 extern int signgam;
```

63651 **DESCRIPTION**63652 Refer to *lgamma()*.

63653 **NAME**

63654 sigpause — remove a signal from the signal mask and suspend the thread

63655 **SYNOPSIS**

```
63656 OB XSI #include <signal.h>  
63657 int sigpause(int sig);
```

63658 **DESCRIPTION**

63659 Refer to *sighold()*.

63660 **NAME**

63661 sigpending ‡examine pending signals

63662 **SYNOPSIS**

```
63663 CX #include <signal.h>
63664 int sigpending(sigset_t *set);
```

63665 **DESCRIPTION**

63666 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of
63667 signals that are blocked from delivery to the calling thread and that are pending on the process
63668 or the calling thread.

63669 **RETURN VALUE**

63670 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and
63671 *errno* set to indicate the error.

63672 **ERRORS**

63673 No errors are defined.

63674 **EXAMPLES**

63675 None.

63676 **APPLICATION USAGE**

63677 None.

63678 **RATIONALE**

63679 None.

63680 **FUTURE DIRECTIONS**

63681 None.

63682 **SEE ALSO**63683 *exec*, *pthread_sigmask()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*

63684 XBD <signal.h>

63685 **CHANGE HISTORY**

63686 First released in Issue 3.

63687 **Issue 5**

63688 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

63689 **Issue 6**

63690 The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an
63691 extension over the ISO C standard.

63692 **NAME**

63693 sigprocmask — examine and change blocked signals

63694 **SYNOPSIS**

```
63695 CX #include <signal.h>
63696 int sigprocmask(int how, const sigset_t *restrict set,
63697 sigset_t *restrict oset);
```

63698 **DESCRIPTION**

63699 Refer to [pthread_sigmask\(\)](#).

63700 **NAME**

63701 sigqueue — queue a signal to a process

63702 **SYNOPSIS**

```
63703 CX #include <signal.h>
63704 int sigqueue(pid_t pid, int signo, union sigval value);
```

63705 **DESCRIPTION**

63706 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value
 63707 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking
 63708 is performed but no signal is actually sent. The null signal can be used to check the validity of
 63709 *pid*.

63710 The conditions required for a process to have permission to queue a signal to another process are
 63711 the same as for the *kill()* function.

63712 The *sigqueue()* function shall return immediately. If SA_SIGINFO is set for *signo* and if the
 63713 resources were available to queue the signal, the signal shall be queued and sent to the receiving
 63714 process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving
 63715 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this
 63716 call.

63717 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked
 63718 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*
 63719 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the
 63720 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the
 63721 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.
 63722 The selection order between realtime and non-realtime signals, or between multiple pending
 63723 non-realtime signals, is unspecified.

63724 **RETURN VALUE**

63725 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*
 63726 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set
 63727 *errno* to indicate the error.

63728 **ERRORS**

63729 The *sigqueue()* function shall fail if:

63730 [EAGAIN] No resources are available to queue the signal. The process has already
 63731 queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or
 63732 a system-wide resource limit has been exceeded.

63733 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

63734 [EPERM] The process does not have appropriate privileges to send the signal to the
 63735 receiving process.

63736 [ESRCH] The process *pid* does not exist.

63737 **EXAMPLES**

63738 None.

63739 **APPLICATION USAGE**

63740 None.

63741 **RATIONALE**

63742 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another
 63743 process, specifying the application-defined value. This is common practice in realtime
 63744 applications on existing realtime systems. It was felt that specifying another function in the
 63745 *sig...* name space already carved out for signals was preferable to extending the interface to
 63746 *kill()*.

63747 Such a function became necessary when the put/get event function of the message queues was
 63748 removed. It should be noted that the *sigqueue()* function implies reduced performance in a
 63749 security-conscious implementation as the access permissions between the sender and receiver
 63750 have to be checked on each send when the *pid* is resolved into a target process. Such access
 63751 checks were necessary only at message queue open in the previous interface.

63752 The standard developers required that *sigqueue()* have the same semantics with respect to the
 63753 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty
 63754 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the
 63755 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function
 63756 queues a signal to a single process specified by the *pid* argument.

63757 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An
 63758 explicit limit on the number of queued signals that a process could send was introduced. While
 63759 the limit is “per-sender”, this volume of POSIX.1-2017 does not specify that the resources be part
 63760 of the state of the sender. This would require either that the sender be maintained after exit until
 63761 all signals that it had sent to other processes were handled or that all such signals that had not
 63762 yet been acted upon be removed from the queue(s) of the receivers. This volume of
 63763 POSIX.1-2017 does not preclude this behavior, but an implementation that allocated queuing
 63764 resources from a system-wide pool (with per-sender limits) and that leaves queued signals
 63765 pending after the sender exits is also permitted.

63766 **FUTURE DIRECTIONS**

63767 None.

63768 **SEE ALSO**63769 [Section 2.8.1](#) (on page 503)63770 XBD [<signal.h>](#)63771 **CHANGE HISTORY**

63772 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
 63773 POSIX Threads Extension.

63774 **Issue 6**63775 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.

63776 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 63777 implementation does not support the Realtime Signals Extension option.

63778 **Issue 7**63779 The *sigqueue()* function is moved from the Realtime Signals Extension option to the Base.

63780 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0332 [844] is applied.

63781 **NAME**

63782 sigrelse, sigset — signal management

63783 **SYNOPSIS**

```
63784 OB XSI #include <signal.h>
63785 int sigrelse(int sig);
63786 void (*sigset(int sig, void (*disp)(int)))(int);
```

63787 **DESCRIPTION**63788 Refer to *sighold()*.

63789 **NAME**

63790 sigsetjmp ‡set jump point for a non-local goto

63791 **SYNOPSIS**

```
63792 CX #include <setjmp.h>
63793 int sigsetjmp(sigjmp_buf env, int savemask);
```

63794 **DESCRIPTION**63795 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:63796 References to *setjmp()* are equivalent to *sigsetjmp()*.63797 References to *longjmp()* are equivalent to *siglongjmp()*.63798 If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.63800 **RETURN VALUE**63801 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.63803 **ERRORS**

63804 No errors are defined.

63805 **EXAMPLES**

63806 None.

63807 **APPLICATION USAGE**63808 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.63810 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified whether the signal mask is saved.63812 **RATIONALE**63813 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to permit implementors to recognize the name in the compiler and not implement an actual function. These same restrictions apply to the *sigsetjmp()* macro.

63816 There are processors that cannot easily support these calls, but this was not considered a sufficient reason to exclude them.

63818 4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *_setjmp()* and *_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal masks, while *_setjmp()* and *_longjmp()* do not. On System V Release 3 and in corresponding issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to signal masks is not specified, and a new set of functions is defined instead.63825 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts saved by other threads would be at best a questionable practice and were not considered worthy of standardization.

63829 **FUTURE DIRECTIONS**

63830 None.

63831 **SEE ALSO**63832 *pthread_sigmask()*, *siglongjmp()*, *signal()*, *sigsuspend()*63833 XBD **<setjmp.h>**63834 **CHANGE HISTORY**

63835 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63836 **Issue 5**

63837 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

63838 **Issue 6**63839 The DESCRIPTION is reworded in terms of *setjmp()*.63840 The SYNOPSIS is marked CX since the presence of this function in the **<setjmp.h>** header is an
63841 extension over the ISO C standard.

63842 **NAME**

63843 sigsuspend ‡wait for a signal

63844 **SYNOPSIS**

```
63845 CX #include <signal.h>
63846 int sigsuspend(const sigset_t *sigmask);
```

63847 **DESCRIPTION**

63848 The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set
 63849 of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose
 63850 action is either to execute a signal-catching function or to terminate the process. This shall not
 63851 cause any other signals that may have been pending on the process to become pending on the
 63852 thread.

63853 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to
 63854 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching
 63855 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*
 63856 call.

63857 It is not possible to block signals that cannot be ignored. This is enforced by the system without
 63858 causing an error to be indicated.

63859 **RETURN VALUE**

63860 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion
 63861 return value. If a return occurs, -1 shall be returned and *errno* set to indicate the error.

63862 **ERRORS**

63863 The *sigsuspend()* function shall fail if:

63864 [EINTR] A signal is caught by the calling process and control is returned from the
 63865 signal-catching function.

63866 **EXAMPLES**

63867 None.

63868 **APPLICATION USAGE**

63869 Normally, at the beginning of a critical code section, a specified set of signals is blocked using
 63870 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait
 63871 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was
 63872 returned by the *sigprocmask()* call.

63873 **RATIONALE**

63874 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers
 63875 can install an additional cancellation handler which resets the signal mask to the expected value.

```
63876 void cleanup(void *arg)
63877 {
63878     sigset_t *ss = (sigset_t *) arg;
63879     pthread_sigmask(SIG_SETMASK, ss, NULL);
63880 }
63881 int call_sigsuspend(const sigset_t *mask)
63882 {
63883     sigset_t oldmask;
63884     int result;
63885     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
63886     pthread_cleanup_push(cleanup, &oldmask);
```

```
63887         result = sigsuspend(sigmask);
63888         pthread_cleanup_pop(0);
63889         return result;
63890     }
```

63891 FUTURE DIRECTIONS

63892 None.

63893 SEE ALSO

63894 [Section 2.4](#) (on page 488), [pause\(\)](#), [sigaction\(\)](#), [sigaddset\(\)](#), [sigdelset\(\)](#), [sigemptyset\(\)](#), [sigfillset\(\)](#)

63895 XBD [<signal.h>](#)

63896 CHANGE HISTORY

63897 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

63898 Issue 5

63899 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

63900 Issue 6

63901 The text in the RETURN VALUE section has been changed from “suspends process execution”
63902 to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

63903 Text in the APPLICATION USAGE section has been replaced.

63904 The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an
63905 extension over the ISO C standard.

63906 Issue 7

63907 SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

63908 **NAME**

63909 sigtimedwait, sigwaitinfo ‡wait for queued signals

63910 **SYNOPSIS**

```
63911 CX #include <signal.h>
63912 int sigtimedwait(const sigset_t *restrict set,
63913 siginfo_t *restrict info,
63914 const struct timespec *restrict timeout);
63915 int sigwaitinfo(const sigset_t *restrict set,
63916 siginfo_t *restrict info);
```

63917 **DESCRIPTION**

63918 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals
63919 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the
63920 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-
63921 valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall return
63922 immediately with an error. If *timeout* is the null pointer, the behavior is unspecified. If the
63923 Monotonic Clock option is supported, the **CLOCK_MONOTONIC** clock shall be used to
63924 measure the time interval specified by the *timeout* argument.

63925 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of
63926 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the
63927 lowest numbered one. The selection order between realtime and non-realtime signals, or
63928 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at
63929 the time of the call, the calling thread shall be suspended until one or more signals in *set* become
63930 pending or until it is interrupted by an unblocked, caught signal.

63931 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function, except that the return
63932 value and the error reporting method are different (see RETURN VALUE), and that if the *info*
63933 argument is non-NULL, the selected signal number shall be stored in the *si_signo* member, and
63934 the cause of the signal shall be stored in the *si_code* member. If any value is queued to the
63935 selected signal, the first such queued value shall be dequeued and, if the *info* argument is non-
63936 NULL, the value shall be stored in the *si_value* member of *info*. The system resource used to
63937 queue the signal shall be released and returned to the system for other use. If no value is
63938 queued, the content of the *si_value* member is undefined. If no further signals are queued for the
63939 selected signal, the pending indication for that signal shall be reset.

63940 **RETURN VALUE**

63941 Upon successful completion (that is, one of the signals specified by *set* is pending or is
63942 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,
63943 the function shall return a value of -1 and set *errno* to indicate the error.

63944 **ERRORS**

63945 The *sigtimedwait()* function shall fail if:

63946 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.

63947 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

63948 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be
63949 documented in system documentation whether this error causes these
63950 functions to fail.

63951 The *sigtimedwait()* function may also fail if:
 63952 [EINVAL] The *timeout* argument specified a *tv_nsec* value less than zero or greater than
 63953 or equal to 1 000 million.

63954 An implementation should only check for this error if no signal is pending in *set* and it is
 63955 necessary to wait.

63956 EXAMPLES

63957 None.

63958 APPLICATION USAGE

63959 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application developers
 63960 should note that this is inconsistent with other functions such as *pthread_cond_timedwait()* that
 63961 return [ETIMEDOUT].

63962 Note that in order to ensure that generated signals are queued and signal values passed to
 63963 *sigqueue()* are available in *si_value*, applications which use *sigwaitinfo()* or *sigtimedwait()* need to
 63964 set the SA_SIGINFO flag for each signal in the set (see Section 2.4, on page 488). This means
 63965 setting each signal to be handled by a three-argument signal-catching function, even if the
 63966 handler will never be called. It is not possible (portably) to set a signal handler to SIG_DFL
 63967 while setting the SA_SIGINFO flag, because assigning to the *sa_handler* member of **struct**
 63968 **sigaction** instead of the *sa_sigaction* member would result in undefined behavior, and SIG_DFL
 63969 need not be assignment-compatible with *sa_sigaction*. Even if an assignment of SIG_DFL to
 63970 *sa_sigaction* is accepted by the compiler, the implementation need not treat this value as special—
 63971 it could just be taken as the address of a signal-catching function.

63972 RATIONALE

63973 Existing programming practice on realtime systems uses the ability to pause waiting for a
 63974 selected set of events and handle the first event that occurs in-line instead of in a signal-handling
 63975 function. This allows applications to be written in an event-directed style similar to a state
 63976 machine. This style of programming is useful for largescale transaction processing in which the
 63977 overall throughput of an application and the ability to clearly track states are more important
 63978 than the ability to minimize the response time of individual event handling.

63979 It is possible to construct a signal-waiting macro function out of the realtime signal function
 63980 mechanism defined in this volume of POSIX.1-2017. However, such a macro has to include the
 63981 definition of a generalized handler for all signals to be waited on. A significant portion of the
 63982 overhead of handler processing can be avoided if the signal-waiting function is provided by the
 63983 kernel. This volume of POSIX.1-2017 therefore provides two signal-waiting functions — one that
 63984 waits indefinitely and one with a timeout — as part of the overall realtime signal function
 63985 specification.

63986 The specification of a function with a timeout allows an application to be written that can be
 63987 broken out of a wait after a set period of time if no event has occurred. It was argued that setting
 63988 a timer event before the wait and recognizing the timer event in the wait would also implement
 63989 the same functionality, but at a lower performance level. Because of the performance
 63990 degradation associated with the user-level specification of a timer event and the subsequent
 63991 cancellation of that timer event after the wait completes for a valid event, and the complexity
 63992 associated with handling potential race conditions associated with the user-level method, the
 63993 separate function has been included.

63994 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*
 63995 function defined by this volume of POSIX.1-2017. The only difference is that *sigwaitinfo()* returns
 63996 the queued signal value in the *value* argument. The return of the queued value is required so that
 63997 applications can differentiate between multiple events queued to the same signal number.

63998 The two distinct functions are being maintained because some implementations may choose to
63999 implement the POSIX Threads Extension functions and not implement the queued signals
64000 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*
64001 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a
64002 macro on *sigwaitinfo()*.

64003 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns
64004 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed
64005 wait, and immediate return, and concerns regarding consistency with other functions where the
64006 conditional and timed waits were separate functions from the pure blocking function. The
64007 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a macro
64008 with a null pointer for *timeout*.

64009 The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-
64010 generated signals. One important question was how many threads that are suspended in a call
64011 to a *sigwait()* function for a signal should return from the call when the signal is sent. Four
64012 choices were considered:

- 64013 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 64014 2. One or more threads return.
- 64015 3. All waiting threads return.
- 64016 4. Exactly one thread returns.

64017 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The
64018 “one or more” behavior made implementation of conforming packages easy at the expense of
64019 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in
64020 application code in order to achieve predictable behavior. There was concern that the “all
64021 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU
64022 resources by replicating the signals in the general case. Furthermore, no convincing examples
64023 could be presented that delivery to all was either simpler or more powerful than delivery to one.

64024 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*
64025 function for a signal should return when that signal occurs. This is not an onerous restriction as:

64026 A multi-way signal wait can be built from the single-way wait.

64027 Signals should only be handled by application-level code, as library routines cannot guess
64028 what the application wants to do with signals generated for the entire process.

64029 Applications can thus arrange for a single thread to wait for any given signal and call any
64030 needed routines upon its arrival.

64031 In an application that is using signals for interprocess communication, signal processing is
64032 typically done in one place. Alternatively, if the signal is being caught so that process cleanup
64033 can be done, the signal handler thread can call separate process cleanup routines for each
64034 portion of the application. Since the application main line started each portion of the application,
64035 it is at the right abstraction level to tell each portion of the application to clean up.

64036 Certainly, there exist programming styles where it is logical to consider waiting for a single
64037 signal in multiple threads. A simple *sigwait_multiple()* routine can be constructed to achieve this
64038 goal. A possible implementation would be to have each *sigwait_multiple()* caller registered as
64039 having expressed interest in a set of signals. The caller then waits on a thread-specific condition
64040 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.
64041 When the *sigwait()* function returns, the appropriate state is set and condition variables are
64042 broadcast. New *sigwait_multiple()* callers may cause the pending *sigwait()* call to be canceled

64043 and reissued in order to update the set of signals being waited for.

64044 **FUTURE DIRECTIONS**

64045 None.

64046 **SEE ALSO**

64047 [Section 2.4](#) (on page 488), [Section 2.8.1](#) (on page 503), [pause\(\)](#), [pthread_sigmask\(\)](#), [sigaction\(\)](#),
64048 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigwait\(\)](#)

64049 XBD [<signal.h>](#), [<time.h>](#)

64050 **CHANGE HISTORY**

64051 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
64052 POSIX Threads Extension.

64053 **Issue 6**

64054 These functions are marked as part of the Realtime Signals Extension option.

64055 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function
64056 has been corrected so that the second argument is of type **siginfo_t** *.

64057 The [ENOSYS] error condition has been removed as stubs need not be provided if an
64058 implementation does not support the Realtime Signals Extension option.

64059 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
64060 CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.

64061 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment
64062 with the ISO/IEC 9899:1999 standard.

64063 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/130 is applied, restoring wording in the
64064 RETURN VALUE section to that in the original base document (“An implementation should
64065 only check for this error if no signal is pending in *set* and it is necessary to wait”).

64066 **Issue 7**

64067 The *sigtimedwait()* and *sigwaitinfo()* functions are moved from the Realtime Signals Extension
64068 option to the Base.

64069 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0583 [392] is applied.

64070 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0333 [815] is applied.

64071 **NAME**

64072 sigwait ‡wait for queued signals

64073 **SYNOPSIS**

```
64074 CX #include <signal.h>
64075 int sigwait(const sigset_t *restrict set, int *restrict sig);
```

64076 **DESCRIPTION**

64077 The *sigwait()* function shall select a pending signal from *set*, atomically clear it from the system's
 64078 set of pending signals, and return that signal number in the location referenced by *sig*. If prior to
 64079 the call to *sigwait()* there are multiple pending instances of a single signal number, it is
 64080 implementation-defined whether upon successful return there are any remaining pending
 64081 signals for that signal number. If the implementation supports queued signals and there are
 64082 multiple signals queued for the signal number selected, the first such queued signal shall cause a
 64083 return from *sigwait()* and the remainder shall remain queued. If no signal in *set* is pending at the
 64084 time of the call, the thread shall be suspended until one or more becomes pending. The signals
 64085 defined by *set* shall have been blocked at the time of the call to *sigwait()*; otherwise, the behavior
 64086 is undefined. The effect of *sigwait()* on the signal actions for the signals in *set* is unspecified.

64087 If more than one thread is using *sigwait()* to wait for the same signal, no more than one of these
 64088 threads shall return from *sigwait()* with the signal number. If more than a single thread is
 64089 blocked in *sigwait()* for a signal when that signal is generated for the process, it is unspecified
 64090 which of the waiting threads returns from *sigwait()*. If the signal is generated for a specific
 64091 thread, as by *pthread_kill()*, only that thread shall return.

64092 Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it
 64093 shall be the lowest numbered one. The selection order between realtime and non-realtime
 64094 signals, or between multiple pending non-realtime signals, is unspecified.

64095 **RETURN VALUE**

64096 Upon successful completion, *sigwait()* shall store the signal number of the received signal at the
 64097 location referenced by *sig* and return zero. Otherwise, an error number shall be returned to
 64098 indicate the error.

64099 **ERRORS**

64100 The *sigwait()* function may fail if:

64101 [EINVAL] The *set* argument contains an invalid or unsupported signal number.

64102 **EXAMPLES**

64103 None.

64104 **APPLICATION USAGE**

64105 None.

64106 **RATIONALE**

64107 To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-2017
 64108 provides the *sigwait()* function. For most cases where a thread has to wait for a signal, the
 64109 *sigwait()* function should be quite convenient, efficient, and adequate.

64110 However, requests were made for a lower-level primitive than *sigwait()* and for semaphores that
 64111 could be used by threads. After some consideration, threads were allowed to use semaphores
 64112 and *sem_post()* was defined to be async-signal-safe.

64113 In summary, when it is necessary for code run in response to an asynchronous signal to notify a
 64114 thread, *sigwait()* should be used to handle the signal. Alternatively, if the implementation
 64115 provides semaphores, they also can be used, either following *sigwait()* or from within a signal

64116 handling routine previously registered with *sigaction()*.

64117 **FUTURE DIRECTIONS**

64118 None.

64119 **SEE ALSO**

64120 [Section 2.4](#) (on page 488), [Section 2.8.1](#) (on page 503), [pause\(\)](#), [pthread_sigmask\(\)](#), [sigaction\(\)](#),
64121 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigtimedwait\(\)](#)

64122 XBD [<signal.h>](#), [<time.h>](#)

64123 **CHANGE HISTORY**

64124 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
64125 POSIX Threads Extension.

64126 **Issue 6**

64127 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the
64128 ISO/IEC 9899:1999 standard.

64129 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/131 is applied, updating the
64130 DESCRIPTION to state that if more than a single thread is blocked in *sigwait()*, it is unspecified
64131 which of the waiting threads returns, and that if a signal is generated for a specific thread only
64132 that thread shall return.

64133 **Issue 7**

64134 Functionality relating to the Realtime Signals Extension option is moved to the Base.

64135 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0584 [76] is applied.

64136 **NAME**

64137 sigwaitinfo ‡wait for queued signals

64138 **SYNOPSIS**

```
64139 #include <signal.h>  
64140 int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);
```

64141 **DESCRIPTION**

64142 Refer to *sigtimedwait()*.

64143 **NAME**64144 `sin, sinf, sinl` \dagger 'sine function64145 **SYNOPSIS**

```
64146 #include <math.h>
64147 double sin(double x);
64148 float sinf(float x);
64149 long double sinl(long double x);
```

64150 **DESCRIPTION**

64151 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64152 conflict between the requirements described here and the ISO C standard is unintentional. This
 64153 volume of POSIX.1-2017 defers to the ISO C standard.

64154 These functions shall compute the sine of their argument x , measured in radians.

64155 An application wishing to check for error situations should set *errno* to zero and call
 64156 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 64157 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 64158 zero, an error has occurred.

64159 **RETURN VALUE**

64160 Upon successful completion, these functions shall return the sine of x .

64161 MX If x is NaN, a NaN shall be returned.

64162 If x is ± 0 , x shall be returned.

64163 If x is subnormal, a range error may occur

64164 MXX and x should be returned.

64165 MX If x is not returned, *sin()*, *sinf()*, and *sinl()* shall return an implementation-defined value no
 64166 greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

64167 If x is $\pm\text{Inf}$, a domain error shall occur, and a NaN shall be returned.

64168 **ERRORS**

64169 These functions shall fail if:

64170 MX **Domain Error** The x argument is $\pm\text{Inf}$.

64171 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 64172 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 64173 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 64174 shall be raised.

64175 These functions may fail if:

64176 MX **Range Error** The value of x is subnormal

64177 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 64178 then *errno* shall be set to [ERANGE]. If the integer expression
 64179 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 64180 floating-point exception shall be raised.

64181 **EXAMPLES**64182 **Taking the Sine of a 45-Degree Angle**

```

64183 #include <math.h>
64184 ...
64185 double radians = 45.0 * M_PI / 180;
64186 double result;
64187 ...
64188 result = sin(radians);

```

64189 **APPLICATION USAGE**

64190 These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

64191 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 64192 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

64193 **RATIONALE**

64194 None.

64195 **FUTURE DIRECTIONS**

64196 None.

64197 **SEE ALSO**

64198 [asin\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

64199 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

64200 **CHANGE HISTORY**

64201 First released in Issue 1. Derived from Issue 1 of the SVID.

64202 **Issue 5**

64203 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
 64204 in previous issues.

64205 **Issue 6**

64206 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

64207 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 64208 revised to align with the ISO/IEC 9899:1999 standard.

64209 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 64210 marked.

64211 **Issue 7**

64212 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0585 [68] and XSH/TC1-2008/0586
 64213 [320] are applied.

64214 **NAME**

64215 sinh, sinhf, sinhl ‡hyperbolic sine functions

64216 **SYNOPSIS**

64217 #include <math.h>

64218 double sinh(double x);

64219 float sinhf(float x);

64220 long double sinh1(long double x);

64221 **DESCRIPTION**64222 CX The functionality described on this reference page is aligned with the ISO C standard. Any
64223 conflict between the requirements described here and the ISO C standard is unintentional. This
64224 volume of POSIX.1-2017 defers to the ISO C standard.64225 These functions shall compute the hyperbolic sine of their argument *x*.64226 An application wishing to check for error situations should set *errno* to zero and call
64227 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
64228 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
64229 zero, an error has occurred.64230 **RETURN VALUE**64231 Upon successful completion, these functions shall return the hyperbolic sine of *x*.64232 If the result would cause an overflow, a range error shall occur and \pm HUGE_VAL,
64233 \pm HUGE_VALF, and \pm HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for
64234 the type of the function.64235 MX If *x* is NaN, a NaN shall be returned.64236 If *x* is ± 0 or \pm Inf, *x* shall be returned.64237 If *x* is subnormal, a range error may occur64238 MXX and *x* should be returned.64239 MX If *x* is not returned, *sinh*(*x*), *sinhf*(*x*), and *sinh1*(*x*) shall return an implementation-defined value no
64240 greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.64241 **ERRORS**

64242 These functions shall fail if:

64243 Range Error The result would cause an overflow.

64244 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
64245 then *errno* shall be set to [ERANGE]. If the integer expression
64246 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
64247 floating-point exception shall be raised.

64248 These functions may fail if:

64249 MX Range Error The value *x* is subnormal.64250 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
64251 then *errno* shall be set to [ERANGE]. If the integer expression
64252 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
64253 floating-point exception shall be raised.

64254 **EXAMPLES**

64255 None.

64256 **APPLICATION USAGE**

64257 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
64258 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

64259 **RATIONALE**

64260 None.

64261 **FUTURE DIRECTIONS**

64262 None.

64263 **SEE ALSO**64264 [*asinh\(\)*](#), [*cosh\(\)*](#), [*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*isnan\(\)*](#), [*tanh\(\)*](#)64265 XBD [Section 4.20](#) (on page 117), [<math.h>](#)64266 **CHANGE HISTORY**

64267 First released in Issue 1. Derived from Issue 1 of the SVID.

64268 **Issue 5**

64269 The DESCRIPTION is updated to indicate how an application should check for an error. This
64270 text was previously published in the APPLICATION USAGE section.

64271 **Issue 6**64272 The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

64273 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
64274 revised to align with the ISO/IEC 9899:1999 standard.

64275 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
64276 marked.

64277 **Issue 7**

64278 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0587 [68] is applied.

64279 **NAME**

64280 sinl †'sine function

64281 **SYNOPSIS**

64282 #include <math.h>

64283 long double sinl(long double x);

64284 **DESCRIPTION**

64285 Refer to *sin()*.

64286 **NAME**

64287 sleep †suspend execution for an interval of time

64288 **SYNOPSIS**

64289 #include <unistd.h>

64290 unsigned sleep(unsigned seconds);

64291 **DESCRIPTION**

64292 The *sleep()* function shall cause the calling thread to be suspended from execution until either
 64293 the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is
 64294 delivered to the calling thread and its action is to invoke a signal-catching function or to
 64295 terminate the process. The suspension time may be longer than requested due to the scheduling
 64296 of other activity by the system.

64297 In single-threaded programs, *sleep()* may make use of SIGALRM. In multi-threaded programs,
 64298 *sleep()* shall not make use of SIGALRM and the remainder of this DESCRIPTION does not apply.

64299 If a SIGALRM signal is generated for the calling process during execution of *sleep()* and if the
 64300 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep()*
 64301 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also
 64302 unspecified whether it remains pending after *sleep()* returns or it is discarded.

64303 If a SIGALRM signal is generated for the calling process during execution of *sleep()*, except as a
 64304 result of a prior call to *alarm()*, and if the SIGALRM signal is not being ignored or blocked from
 64305 delivery, it is unspecified whether that signal has any effect other than causing *sleep()* to return.

64306 If a signal-catching function interrupts *sleep()* and examines or changes either the time a
 64307 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or
 64308 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

64309 If a signal-catching function interrupts *sleep()* and calls *siglongjmp()* or *longjmp()* to restore an
 64310 environment saved prior to the *sleep()* call, the action associated with the SIGALRM signal and
 64311 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also
 64312 unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is
 64313 restored as part of the environment.

64314 XSI Interactions between *sleep()* and *setitimer()* are unspecified.

64315 **RETURN VALUE**

64316 If *sleep()* returns because the requested time has elapsed, the value returned shall be 0. If *sleep()*
 64317 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested
 64318 time minus the time actually slept) in seconds.

64319 **ERRORS**

64320 No errors are defined.

64321 **EXAMPLES**

64322 None.

64323 **APPLICATION USAGE**

64324 None.

64325 **RATIONALE**

64326 There are two general approaches to the implementation of the *sleep()* function. One is to use the
 64327 *alarm()* function to schedule a SIGALRM signal and then suspend the calling thread waiting for
 64328 that signal. The other is to implement an independent facility. This volume of POSIX.1-2017
 64329 permits either approach in single-threaded programs, but the simple alarm/suspend
 64330 implementation is not appropriate for multi-threaded programs.

64331 In order to comply with the requirement that no primitive shall change a process attribute unless
 64332 explicitly described by this volume of POSIX.1-2017, an implementation using SIGALRM must
 64333 carefully take into account any SIGALRM signal scheduled by previous *alarm()* calls, the action
 64334 previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has
 64335 been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened to
 64336 that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-
 64337 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*
 64338 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The
 64339 action and blocking for SIGALRM must be saved and restored.

64340 Historical implementations often implement the SIGALRM-based version using *alarm()* and
 64341 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*.
 64342 Another such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid
 64343 that window. That implementation introduces a different problem: when the SIGALRM signal
 64344 interrupts a signal-catching function installed by the user to catch a different signal, the
 64345 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,
 64346 *alarm()*, and *sigsuspend()* can avoid these problems.

64347 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,
 64348 differences between the two types of implementations. These are the cases mentioned in this
 64349 volume of POSIX.1-2017 where some other activity relating to SIGALRM takes place, and the
 64350 results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of
 64351 concern to most applications.

64352 See also the discussion of the term *realtime* in *alarm()*.

64353 Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early
 64354 under *alarm()* applies to *sleep()* as well.

64355 Application developers should note that the type of the argument *seconds* and the return value of
 64356 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application
 64357 cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the
 64358 ISO C standard sets as 65535, and any application passing a larger value is restricting its
 64359 portability. A different type was considered, but historical implementations, including those
 64360 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

64361 Scheduling delays may cause the process to return from the *sleep()* function significantly after
 64362 the requested time. In such cases, the return value should be set to zero, since the formula
 64363 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an
 64364 **unsigned**.

64365 FUTURE DIRECTIONS

64366 A future version of this standard may require that *sleep()* does not make use of SIGALRM in all
 64367 programs, not just multi-threaded programs.

64368 SEE ALSO

64369 *alarm()*, *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*

64370 XBD <**unistd.h**>

64371 CHANGE HISTORY

64372 First released in Issue 1. Derived from Issue 1 of the SVID.

64373 Issue 5

64374 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

64375 **Issue 6**

64376 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/132 is applied, making a correction in the
64377 RATIONALE section.

64378 **Issue 7**

64379 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0334 [625] is applied.

64380 **NAME**

64381 sprintf †'print formatted output

64382 **SYNOPSIS**

64383 #include <stdio.h>

64384 int snprintf(char *restrict *s*, size_t *n*,64385 const char *restrict *format*, ...);64386 **DESCRIPTION**64387 Refer to *fprintf()*.

64388 **NAME**

64389 socketmark ¶determine whether a socket is at the out-of-band mark

64390 **SYNOPSIS**

64391 #include <sys/socket.h>

64392 int socketmark(int s);

64393 **DESCRIPTION**

64394 The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at the out-of-band data mark (see [Section 2.10.12](#), on page 527). If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, the *socketmark()* function shall return 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. The *socketmark()* function shall not remove the mark from the stream.

64400 **RETURN VALUE**

64401 Upon successful completion, the *socketmark()* function shall return a value indicating whether the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data preceding the mark has been read, the return value shall be 1; if there is no mark, or if data precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it shall return a value of -1 and set *errno* to indicate the error.

64406 **ERRORS**64407 The *socketmark()* function shall fail if:64408 [EBADF] The *s* argument is not a valid file descriptor.64409 [ENOTTY] The file associated with the *s* argument is not a socket.64410 **EXAMPLES**

64411 None.

64412 **APPLICATION USAGE**

64413 The use of this function between receive operations allows an application to determine which received data precedes the out-of-band data and which follows the out-of-band data.

64415 There is an inherent race condition in the use of this function. On an empty receive queue, the current read of the location might well be at the “mark”, but the system has no way of knowing that the next data segment that will arrive from the network will carry the mark, and *socketmark()* will return false, and the next read operation will silently consume the mark.

64419 Hence, this function can only be used reliably when the application already knows that the out-of-band data has been seen by the system or that it is known that there is data waiting to be read at the socket (via SIGURG or *select()*). See [Section 2.10.11](#) (on page 526), [Section 2.10.12](#) (on page 527), [Section 2.10.14](#) (on page 527), and *pselect()* for details.

64423 **RATIONALE**

64424 The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which implemented the same functionality on many implementations. Using a wrapper function follows the adopted conventions to avoid specifying commands to the *ioctl()* function, other than those now included to support XSI STREAMS. The *socketmark()* function could be implemented as follows:

64429 #include <sys/ioctl.h>

64430 int socketmark(int s)

64431 {

64432 int val;

```
64433         if (ioctl(s,SIOCATMARK,&val)==-1)
64434             return(-1);
64435         return(val);
64436     }
```

64437 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of
64438 SIOCATMARK.

64439 **FUTURE DIRECTIONS**

64440 None.

64441 **SEE ALSO**

64442 [Section 2.10.12](#) (on page 527), [pselect\(\)](#), [recv\(\)](#), [recvmsg\(\)](#)

64443 XBD [<sys/socket.h>](#)

64444 **CHANGE HISTORY**

64445 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

64446 **Issue 7**

64447 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

64448 **NAME**

64449 socket — create an endpoint for communication

64450 **SYNOPSIS**

64451 #include <sys/socket.h>

64452 int socket(int *domain*, int *type*, int *protocol*);64453 **DESCRIPTION**

64454 The *socket()* function shall create an unbound socket in a communications domain, and return a
64455 file descriptor that can be used in later function calls that operate on sockets. The file descriptor
64456 shall be allocated as described in [Section 2.14](#) (on page 549).

64457 The *socket()* function takes the following arguments:

64458 *domain* Specifies the communications domain in which a socket is to be created.

64459 *type* Specifies the type of socket to be created.

64460 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*
64461 of 0 causes *socket()* to use an unspecified default protocol appropriate for the
64462 requested socket type.

64463 The *domain* argument specifies the address family used in the communications domain. The
64464 address families supported by the system are implementation-defined.

64465 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>
64466 header.

64467 The *type* argument specifies the socket type, which determines the semantics of communication
64468 over the socket. The following socket types are defined; implementations may specify additional
64469 socket types:

64470 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte streams,
64471 and may provide a transmission mechanism for out-of-band data.

64472 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages of
64473 fixed maximum length.

64474 SOCK_SEQPACKET

64475 Provides sequenced, reliable, bidirectional, connection-mode transmission
64476 paths for records. A record can be sent using one or more output operations
64477 and received using one or more input operations, but a single operation never
64478 transfers part of more than one record. Record boundaries are visible to the
64479 receiver via the MSG_EOR flag.

64480 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
64481 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
64482 be used. The protocols supported by the system are implementation-defined.

64483 The process may need to have appropriate privileges to use the *socket()* function or to create
64484 some sockets.

64485 **RETURN VALUE**

64486 Upon successful completion, *socket()* shall return a non-negative integer, the socket file
64487 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.

64488 ERRORS

64489 The `socket()` function shall fail if:

64490 [EAFNOSUPPORT]

64491 The implementation does not support the specified address family.

64492 [EMFILE] All file descriptors available to the process are currently open.

64493 [ENFILE] No more file descriptors are available for the system.

64494 [EPROTONOSUPPORT]

64495 The protocol is not supported by the address family, or the protocol is not
64496 supported by the implementation.

64497 [EPROTOTYPE] The socket type is not supported by the protocol.

64498 The `socket()` function may fail if:

64499 [EACCES] The process does not have appropriate privileges.

64500 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

64501 [ENOMEM] Insufficient memory was available to fulfill the request.

64502 EXAMPLES

64503 None.

64504 APPLICATION USAGE

64505 The documentation for specific address families specifies which protocols each address family
64506 supports. The documentation for specific protocols specifies which socket types each protocol
64507 supports.

64508 The application can determine whether an address family is supported by trying to create a
64509 socket with *domain* set to the protocol in question.

64510 RATIONALE

64511 None.

64512 FUTURE DIRECTIONS

64513 None.

64514 SEE ALSO

64515 [Section 2.14](#) (on page 549), [accept\(\)](#), [bind\(\)](#), [connect\(\)](#), [getsockname\(\)](#), [getsockopt\(\)](#), [listen\(\)](#), [recv\(\)](#),
64516 [recvfrom\(\)](#), [recvmsg\(\)](#), [send\(\)](#), [sendmsg\(\)](#), [setsockopt\(\)](#), [shutdown\(\)](#), [socketpair\(\)](#)

64517 XBD [<netinet/in.h>](#), [<sys/socket.h>](#)

64518 CHANGE HISTORY

64519 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

64520 Issue 7

64521 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0335 [835] is applied.

64522 **NAME**

64523 socketpair — create a pair of connected sockets

64524 **SYNOPSIS**

```
64525 #include <sys/socket.h>
64526 int socketpair(int domain, int type, int protocol,
64527               int socket_vector[2]);
```

64528 **DESCRIPTION**

64529 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,
 64530 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two
 64531 sockets shall be identical. The file descriptors used in referencing the created sockets shall be
 64532 returned in *socket_vector*[0] and *socket_vector*[1]. The file descriptors shall be allocated as
 64533 described in [Section 2.14](#) (on page 549).

64534 The *socketpair()* function takes the following arguments:

64535	<i>domain</i>	Specifies the communications domain in which the sockets are to be created.
64536	<i>type</i>	Specifies the type of sockets to be created.
64537	<i>protocol</i>	Specifies a particular protocol to be used with the sockets. Specifying a 64538 <i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol 64539 appropriate for the requested socket type.
64540	<i>socket_vector</i>	Specifies a 2-integer array to hold the file descriptors of the created socket pair.

64541 The *type* argument specifies the socket type, which determines the semantics of communications
 64542 over the socket. The following socket types are defined; implementations may specify additional
 64543 socket types:

64544	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte 64545 streams, and may provide a transmission mechanism for out-of-band 64546 data.
64547	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages 64548 of fixed maximum length.
64549	SOCK_SEQPACKET	Provides sequenced, reliable, bidirectional, connection-mode transmission 64550 paths for records. A record can be sent using one or more output 64551 operations and received using one or more input operations, but a single 64552 operation never transfers part of more than one record. Record 64553 boundaries are visible to the receiver via the MSG_EOR flag.

64554 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
 64555 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
 64556 be used. The protocols supported by the system are implementation-defined.

64557 The process may need to have appropriate privileges to use the *socketpair()* function or to create
 64558 some sockets.

64559 **RETURN VALUE**

64560 Upon successful completion, this function shall return 0; otherwise, -1 shall be returned and
 64561 *errno* set to indicate the error, no file descriptors shall be allocated, and the contents of
 64562 *socket_vector* shall be left unmodified.

64563 **ERRORS**64564 The *socketpair()* function shall fail if:

64565 [EAFNOSUPPORT]

64566 The implementation does not support the specified address family.

64567 [EMFILE]

64568 All, or all but one, of the file descriptors available to the process are currently open.

64569 [ENFILE]

No more file descriptors are available for the system.

64570 [EOPNOTSUPP] The specified protocol does not permit creation of socket pairs.

64571 [EPROTONOSUPPORT]

64572 The protocol is not supported by the address family, or the protocol is not

64573 supported by the implementation.

64574 [EPROTOTYPE] The socket type is not supported by the protocol.

64575 The *socketpair()* function may fail if:

64576 [EACCES]

The process does not have appropriate privileges.

64577 [ENOBUFS]

Insufficient resources were available in the system to perform the operation.

64578 [ENOMEM]

Insufficient memory was available to fulfill the request.

64579 **EXAMPLES**

64580 None.

64581 **APPLICATION USAGE**64582 The documentation for specific address families specifies which protocols each address family
64583 supports. The documentation for specific protocols specifies which socket types each protocol
64584 supports.64585 The *socketpair()* function is used primarily with UNIX domain sockets and need not be

64586 supported for other domains.

64587 **RATIONALE**

64588 None.

64589 **FUTURE DIRECTIONS**

64590 None.

64591 **SEE ALSO**64592 [Section 2.14](#) (on page 549), *socket()*64593 XBD [<sys/socket.h>](#)64594 **CHANGE HISTORY**

64595 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

64596 **Issue 7**64597 The description of the [EMFILE] error condition is aligned with the *pipe()* function.

64598 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0336 [835] and XSH/TC2-2008/0337

64599 [483,835] are applied.

64600 **NAME**

64601 sprintf †'print formatted output

64602 **SYNOPSIS**

64603 #include <stdio.h>

64604 int sprintf(char *restrict *s*, const char *restrict *format*, ...);

64605 **DESCRIPTION**

64606 Refer to *fprintf()*.

64607 **NAME**64608 sqrt, sqrtf, sqrtl $\sqrt{\quad}$ root function64609 **SYNOPSIS**

```
64610 #include <math.h>
64611 double sqrt(double x);
64612 float sqrtf(float x);
64613 long double sqrtl(long double x);
```

64614 **DESCRIPTION**

64615 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64616 conflict between the requirements described here and the ISO C standard is unintentional. This
 64617 volume of POSIX.1-2017 defers to the ISO C standard.

64618 These functions shall compute the square root of their argument x , \sqrt{x} .

64619 An application wishing to check for error situations should set *errno* to zero and call
 64620 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 64621 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 64622 zero, an error has occurred.

64623 **RETURN VALUE**

64624 Upon successful completion, these functions shall return the square root of x .

64625 MX For finite values of $x < -0$, a domain error shall occur, and either a NaN (if supported), or an
 64626 implementation-defined value shall be returned.

64627 MX If x is NaN, a NaN shall be returned.

64628 If x is ± 0 or $+\text{Inf}$, x shall be returned.

64629 If x is $-\text{Inf}$, a domain error shall occur, and a NaN shall be returned.

64630 **ERRORS**

64631 These functions shall fail if:

64632 MX Domain Error The finite value of x is < -0 , or x is $-\text{Inf}$.

64633 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 64634 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 64635 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 64636 shall be raised.

64637 **EXAMPLES**64638 **Taking the Square Root of 9.0**

```
64639 #include <math.h>
64640 ...
64641 double x = 9.0;
64642 double result;
64643 ...
64644 result = sqrt(x);
```

64645 **APPLICATION USAGE**

64646 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 64647 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

64648 **RATIONALE**

64649 None.

64650 **FUTURE DIRECTIONS**

64651 None.

64652 **SEE ALSO**64653 *feclearexcept()*, *fetestexcept()*, *isnan()*64654 XBD Section 4.20 (on page 117), `<math.h>`, `<stdio.h>`64655 **CHANGE HISTORY**

64656 First released in Issue 1. Derived from Issue 1 of the SVID.

64657 **Issue 5**64658 The DESCRIPTION is updated to indicate how an application should check for an error. This
64659 text was previously published in the APPLICATION USAGE section.64660 **Issue 6**64661 The *sqrtof()* and *sqrtrl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.64662 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
64663 revised to align with the ISO/IEC 9899:1999 standard.64664 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
64665 marked.64666 **Issue 7**

64667 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0588 [320] is applied.

64668 **NAME**

64669 **rand** †pseudo-random number generator

64670 **SYNOPSIS**

64671 #include <stdlib.h>

64672 void srand(unsigned *seed*);

64673 **DESCRIPTION**

64674 Refer to *rand()*.

64675 **NAME**

64676 srand48 — seed the uniformly distributed double-precision pseudo-random number generator

64677 **SYNOPSIS**

```
64678 XSI       #include <stdlib.h>  
64679           void srand48(long seedval);
```

64680 **DESCRIPTION**

64681 Refer to *drand48()*.

64682 **NAME**

64683 srandom ‡seed pseudo-random number generator

64684 **SYNOPSIS**

```
64685 XSI       #include <stdlib.h>  
64686           void srandom(unsigned seed);
```

64687 **DESCRIPTION**64688 Refer to *initstate()*.

64689 **NAME**

64690 sscanf ‡'convert formatted input

64691 **SYNOPSIS**

64692 #include <stdio.h>

64693 int sscanf(const char *restrict *s*, const char *restrict *format*, ...);

64694 **DESCRIPTION**

64695 Refer to [fscanf\(\)](#).

64696 **NAME**

64697 stat †'get file status

64698 **SYNOPSIS**

64699 #include <sys/stat.h>

64700 int stat(const char *restrict *path*, struct stat *restrict *buf*);

64701 **DESCRIPTION**

64702 Refer to *fstatat()*.

64703 **NAME**

64704 statvfs ‡get file system information

64705 **SYNOPSIS**

64706 #include <sys/statvfs.h>

64707 int statvfs(const char *restrict *path*, struct statvfs *restrict *buf*);

64708 **DESCRIPTION**

64709 Refer to [fstatvfs\(\)](#).

64710 **NAME**

64711 stderr, stdin, stdout ¶standard I/O streams

64712 **SYNOPSIS**

64713 #include <stdio.h>

64714 extern FILE *stderr, *stdin, *stdout;

64715 **DESCRIPTION**

64716 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64717 conflict between the requirements described here and the ISO C standard is unintentional. This
 64718 volume of POSIX.1-2017 defers to the ISO C standard.

64719 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type
 64720 **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer
 64721 to designate the stream in all further transactions. Normally, there are three open streams with
 64722 constant pointers declared in the <stdio.h> header and associated with the standard open files.

64723 At program start-up, three streams shall be predefined and need not be opened explicitly:
 64724 *standard input* (for reading conventional input), *standard output* (for writing conventional output),
 64725 and *standard error* (for writing diagnostic output). When opened, the standard error stream is not
 64726 fully buffered; the standard input and standard output streams are fully buffered if and only if
 64727 the stream can be determined not to refer to an interactive device.

64728 CX The following symbolic values in <unistd.h> define the file descriptors that shall be associated
 64729 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

64730 STDIN_FILENO Standard input value, *stdin*. Its value is 0.64731 STDOUT_FILENO Standard output value, *stdout*. Its value is 1.64732 STDERR_FILENO Standard error value, *stderr*. Its value is 2.64733 The *stderr* stream is expected to be open for reading and writing.64734 **RETURN VALUE**

64735 None.

64736 **ERRORS**

64737 No errors are defined.

64738 **EXAMPLES**

64739 None.

64740 **APPLICATION USAGE**

64741 None.

64742 **RATIONALE**

64743 None.

64744 **FUTURE DIRECTIONS**

64745 None.

64746 **SEE ALSO**64747 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fprintf()*, *fread()*, *fscanf()*, *fseek()*, *getc()*, *gets()*, *popen()*,
 64748 *putc()*, *puts()*, *read()*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vfprintf()*

64749 XBD <stdio.h>, <unistd.h>

64750 **CHANGE HISTORY**

64751 First released in Issue 1.

64752 **Issue 6**

64753 Extensions beyond the ISO C standard are marked.

64754 A note that *stderr* is expected to be open for reading and writing is added to the DESCRIPTION.

64755 **NAME**

64756 stpcpy — copy a string and return a pointer to the end of the result

64757 **SYNOPSIS**

```
64758 CX #include <string.h>  
64759 char *stpcpy(char *restrict s1, const char *restrict s2);
```

64760 **DESCRIPTION**64761 Refer to *strcpy()*.

64762 **NAME**

64763 stpncpy — copy fixed length string, returning a pointer to the array end

64764 **SYNOPSIS**

```
64765 CX        #include <string.h>
64766            char *stpncpy(char *restrict s1, const char *restrict s2, size_t size);
```

64767 **DESCRIPTION**

64768 Refer to *strncpy()*.

64769 NAME

64770 `strcasecmp`, `strcasecmp_l`, `strncasecmp`, `strncasecmp_l` — case-insensitive string comparisons

64771 SYNOPSIS

```
64772 #include <strings.h>
64773 int strcasecmp(const char *s1, const char *s2);
64774 int strcasecmp_l(const char *s1, const char *s2,
64775                 locale_t locale);
64776 int strncasecmp(const char *s1, const char *s2, size_t n);
64777 int strncasecmp_l(const char *s1, const char *s2,
64778                  size_t n, locale_t locale);
```

64779 DESCRIPTION

64780 The `strcasecmp()` and `strcasecmp_l()` functions shall compare, while ignoring differences in case,
64781 the string pointed to by `s1` to the string pointed to by `s2`. The `strncasecmp()` and `strncasecmp_l()`
64782 functions shall compare, while ignoring differences in case, not more than `n` bytes from the
64783 string pointed to by `s1` to the string pointed to by `s2`.

64784 The `strcasecmp()` and `strncasecmp()` functions use the current locale to determine the case of the
64785 characters.

64786 The `strcasecmp_l()` and `strncasecmp_l()` functions use the locale represented by `locale` to determine
64787 the case of the characters.

64788 When the `LC_CTYPE` category of the locale being used is from the POSIX locale, these functions
64789 shall behave as if the strings had been converted to lowercase and then a byte comparison
64790 performed. Otherwise, the results are unspecified.

64791 The behavior is undefined if the `locale` argument to `strcasecmp_l()` or `strncasecmp_l()` is the special
64792 locale object `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

64793 RETURN VALUE

64794 Upon completion, `strcasecmp()` and `strcasecmp_l()` shall return an integer greater than, equal to,
64795 or less than 0, if the string pointed to by `s1` is, ignoring case, greater than, equal to, or less than
64796 the string pointed to by `s2`, respectively.

64797 Upon successful completion, `strncasecmp()` and `strncasecmp_l()` shall return an integer greater
64798 than, equal to, or less than 0, if the possibly null-terminated array pointed to by `s1` is, ignoring
64799 case, greater than, equal to, or less than the possibly null-terminated array pointed to by `s2`,
64800 respectively.

64801 ERRORS

64802 No errors are defined.

64803 EXAMPLES

64804 None.

64805 APPLICATION USAGE

64806 None.

64807 RATIONALE

64808 None.

64809 FUTURE DIRECTIONS

64810 None.

64811 **SEE ALSO**

64812 [wcscasecmp\(\)](#)

64813 XBD [<strings.h>](#)

64814 **CHANGE HISTORY**

64815 First released in Issue 4, Version 2.

64816 **Issue 5**

64817 Moved from X/OPEN UNIX extension to BASE.

64818 **Issue 7**

64819 The *strcasecmp()* and *strncasecmp()* functions are moved from the XSI option to the Base.

64820 The *strcasecmp_l()* and *strncasecmp_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

64822 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0589 [302], XSH/TC1-2008/0590 [294], XSH/TC1-2008/0591 [283], and XSH/TC1-2008/0592 [283] are applied.

64824 **NAME**

64825 strcat — concatenate two strings

64826 **SYNOPSIS**

64827 #include <string.h>

64828 char *strcat(char *restrict *s1*, const char *restrict *s2*);64829 **DESCRIPTION**

64830 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
64831 conflict between the requirements described here and the ISO C standard is unintentional. This
64832 volume of POSIX.1-2017 defers to the ISO C standard.

64833 The *strcat()* function shall append a copy of the string pointed to by *s2* (including the
64834 terminating NUL character) to the end of the string pointed to by *s1*. The initial byte of *s2*
64835 overwrites the NUL character at the end of *s1*. If copying takes place between objects that
64836 overlap, the behavior is undefined.

64837 **RETURN VALUE**64838 The *strcat()* function shall return *s1*; no return value is reserved to indicate an error.64839 **ERRORS**

64840 No errors are defined.

64841 **EXAMPLES**

64842 None.

64843 **APPLICATION USAGE**

64844 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3
64845 applications. Reliable error detection by this function was never guaranteed.

64846 **RATIONALE**

64847 None.

64848 **FUTURE DIRECTIONS**

64849 None.

64850 **SEE ALSO**64851 [strncat\(\)](#)64852 XBD [<string.h>](#)64853 **CHANGE HISTORY**

64854 First released in Issue 1. Derived from Issue 1 of the SVID.

64855 **Issue 6**64856 The *strcat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

64857 **NAME**

64858 strchr — string scanning operation

64859 **SYNOPSIS**

64860 #include <string.h>

64861 char *strchr(const char *s, int c);

64862 **DESCRIPTION**

64863 CX The functionality described on this reference page is aligned with the ISO C standard. Any
64864 conflict between the requirements described here and the ISO C standard is unintentional. This
64865 volume of POSIX.1-2017 defers to the ISO C standard.

64866 The *strchr()* function shall locate the first occurrence of *c* (converted to a **char**) in the string
64867 pointed to by *s*. The terminating NUL character is considered to be part of the string.

64868 **RETURN VALUE**

64869 Upon completion, *strchr()* shall return a pointer to the byte, or a null pointer if the byte was not
64870 found.

64871 **ERRORS**

64872 No errors are defined.

64873 **EXAMPLES**

64874 None.

64875 **APPLICATION USAGE**

64876 None.

64877 **RATIONALE**

64878 None.

64879 **FUTURE DIRECTIONS**

64880 None.

64881 **SEE ALSO**64882 [strrchr\(\)](#)64883 XBD [<string.h>](#)64884 **CHANGE HISTORY**

64885 First released in Issue 1. Derived from Issue 1 of the SVID.

64886 **Issue 6**

64887 Extensions beyond the ISO C standard are marked.

64888 **NAME**

64889 strcmp — compare two strings

64890 **SYNOPSIS**

64891 #include <string.h>

64892 int strcmp(const char *s1, const char *s2);

64893 **DESCRIPTION**

64894 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64895 conflict between the requirements described here and the ISO C standard is unintentional. This
 64896 volume of POSIX.1-2017 defers to the ISO C standard.

64897 The *strcmp()* function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

64898 The sign of a non-zero return value shall be determined by the sign of the difference between the
 64899 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
 64900 being compared.

64901 **RETURN VALUE**

64902 Upon completion, *strcmp()* shall return an integer greater than, equal to, or less than 0, if the
 64903 string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*,
 64904 respectively.

64905 **ERRORS**

64906 No errors are defined.

64907 **EXAMPLES**64908 **Checking a Password Entry**

64909 The following example compares the information read from standard input to the value of the
 64910 name of the user entry. If the *strcmp()* function returns 0 (indicating a match), a further check
 64911 will be made to see if the user entered the proper old password. The *crypt()* function shall
 64912 encrypt the old password entered by the user, using the value of the encrypted password in the
 64913 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the
 64914 structure, the entered password *oldpasswd* is the correct user's password. Finally, the program
 64915 encrypts the new password so that it can store the information in the **passwd** structure.

```

64916 #include <string.h>
64917 #include <unistd.h>
64918 #include <stdio.h>
64919 ...
64920 int valid_change;
64921 struct passwd *p;
64922 char user[100];
64923 char oldpasswd[100];
64924 char newpasswd[100];
64925 char savepasswd[100];
64926 ...
64927 if (strcmp(p->pw_name, user) == 0) {
64928     if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
64929         strcpy(savepasswd, crypt(newpasswd, user));
64930         p->pw_passwd = savepasswd;
64931         valid_change = 1;
64932     }
64933     else {

```



```
64934         fprintf(stderr, "Old password is not valid\n");
64935     }
64936 }
64937 ...
```

64938 APPLICATION USAGE

64939 None.

64940 RATIONALE

64941 None.

64942 FUTURE DIRECTIONS

64943 None.

64944 SEE ALSO

64945 [strncmp\(\)](#)

64946 XBD [<string.h>](#)

64947 CHANGE HISTORY

64948 First released in Issue 1. Derived from Issue 1 of the SVID.

64949 Issue 6

64950 Extensions beyond the ISO C standard are marked.

64951 **NAME**

64952 strcoll, strcoll_l — string comparison using collating information

64953 **SYNOPSIS**

```
64954 #include <string.h>
64955 int strcoll(const char *s1, const char *s2);
64956 CX int strcoll_l(const char *s1, const char *s2,
64957 locale_t locale);
```

64958 **DESCRIPTION**

64959 CX For `strcoll()`: The functionality described on this reference page is aligned with the ISO C
 64960 standard. Any conflict between the requirements described here and the ISO C standard is
 64961 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

64962 CX The `strcoll()` and `strcoll_l()` functions shall compare the string pointed to by `s1` to the string
 64963 pointed to by `s2`, both interpreted as appropriate to the `LC_COLLATE` category of the current
 64964 CX locale, or of the locale represented by `locale`, respectively.

64965 CX The `strcoll()` and `strcoll_l()` functions shall not change the setting of `errno` if successful.

64966 Since no return value is reserved to indicate an error, an application wishing to check for error
 64967 CX situations should set `errno` to 0, then call `strcoll()`, or `strcoll_l()` then check `errno`.

64968 CX The behavior is undefined if the `locale` argument to `strcoll_l()` is the special locale object
 64969 LC_GLOBAL_LOCALE or is not a valid locale object handle.

64970 **RETURN VALUE**

64971 Upon successful completion, `strcoll()` shall return an integer greater than, equal to, or less than 0,
 64972 according to whether the string pointed to by `s1` is greater than, equal to, or less than the string
 64973 CX pointed to by `s2` when both are interpreted as appropriate to the current locale. On error,
 64974 `strcoll()` may set `errno`, but no return value is reserved to indicate an error.

64975 Upon successful completion, `strcoll_l()` shall return an integer greater than, equal to, or less than
 64976 0, according to whether the string pointed to by `s1` is greater than, equal to, or less than the
 64977 string pointed to by `s2` when both are interpreted as appropriate to the locale represented by
 64978 `locale`. On error, `strcoll_l()` may set `errno`, but no return value is reserved to indicate an error.

64979 **ERRORS**

64980 These functions may fail if:

64981 CX [EINVAL] The `s1` or `s2` arguments contain characters outside the domain of the collating
 64982 sequence.

64983 **EXAMPLES**64984 **Comparing Nodes**

64985 The following example uses an application-defined function, `node_compare()`, to compare two
 64986 nodes based on an alphabetical ordering of the `string` field.

```
64987 #include <string.h>
64988 ...
64989 struct node { /* These are stored in the table. */
64990     char *string;
64991     int length;
64992 };
64993 ...
```

```
64994     int node_compare(const void *node1, const void *node2)
64995     {
64996         return strcoll(((const struct node *)node1)->string,
64997                       ((const struct node *)node2)->string);
64998     }
64999     ...
```

65000 APPLICATION USAGE

65001 The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

65002 RATIONALE

65003 None.

65004 FUTURE DIRECTIONS

65005 None.

65006 SEE ALSO

65007 [alphasort\(\)](#), [strcmp\(\)](#), [strxfrm\(\)](#)

65008 XBD [<string.h>](#)

65009 CHANGE HISTORY

65010 First released in Issue 3.

65011 Issue 5

65012 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

65013 Issue 6

65014 Extensions beyond the ISO C standard are marked.

65015 The following new requirements on POSIX implementations derive from alignment with the
65016 Single UNIX Specification:

65017 The [EINVAL] optional error condition is added.

65018 An example is added.

65019 Issue 7

65020 The *strcoll_1()* function is added from The Open Group Technical Standard, 2006, Extended API
65021 Set Part 4.

65022 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0593 [283] and XSH/TC1-2008/0594
65023 [283] are applied.

65024 **NAME**

65025 stpcpy, strcpy — copy a string and return a pointer to the end of the result

65026 **SYNOPSIS**

65027 #include <string.h>

65028 CX char *stpcpy(char *restrict s1, const char *restrict s2);

65029 char *strcpy(char *restrict s1, const char *restrict s2);

65030 **DESCRIPTION**65031 CX For *strcpy()*: The functionality described on this reference page is aligned with the ISO C
65032 standard. Any conflict between the requirements described here and the ISO C standard is
65033 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.65034 CX The *stpcpy()* and *strcpy()* functions shall copy the string pointed to by *s2* (including the
65035 terminating NUL character) into the array pointed to by *s1*.

65036 If copying takes place between objects that overlap, the behavior is undefined.

65037 **RETURN VALUE**65038 CX The *stpcpy()* function shall return a pointer to the terminating NUL character copied into the *s1*
65039 buffer.65040 The *strcpy()* function shall return *s1*.

65041 No return values are reserved to indicate an error.

65042 **ERRORS**

65043 No errors are defined.

65044 **EXAMPLES**65045 **Construction of a Multi-Part Message in a Single Buffer**

65046 #include <string.h>

65047 #include <stdio.h>

65048 int

65049 main (void)

65050 {

65051 char buffer [10];

65052 char *name = buffer;

65053 name = stpcpy (stpcpy (stpcpy (name, "ice"), "-"), "cream");

65054 puts (buffer);

65055 return 0;

65056 }

65057 **Initializing a String**65058 The following example copies the string "-----" into the *permstring* variable.

65059 #include <string.h>

65060 ...

65061 static char permstring[11];

65062 ...

65063 strcpy(permstring, "-----");

65064 ...

65065 **Storing a Key and Data**

65066 The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the
 65067 key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there.
 65068 (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct**
 65069 **element** *.)

```
65070 #include <string.h>
65071 #include <stdlib.h>
65072 #include <stdio.h>
65073 ...
65074 /* Structure used to read data and store it. */
65075 struct element {
65076     char *key;
65077     char *data;
65078 };
65079 struct element *tbl, *curtbl;
65080 char *key, *data;
65081 int count;
65082 ...
65083 void dbfree(struct element *, int);
65084 ...
65085 if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
65086     perror("malloc"); dbfree(tbl, count); return NULL;
65087 }
65088 strcpy(curtbl->key, key);
65089 if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
65090     perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
65091 }
65092 strcpy(curtbl->data, data);
65093 ...
```

65094 **APPLICATION USAGE**

65095 Character movement is performed differently in different implementations. Thus, overlapping
 65096 moves may yield surprises.

65097 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3
 65098 applications. Reliable error detection by this function was never guaranteed.

65099 **RATIONALE**

65100 None.

65101 **FUTURE DIRECTIONS**

65102 None.

65103 **SEE ALSO**

65104 *strncpy()*, *wcscpy()*

65105 XBD [<string.h>](#)

65106 **CHANGE HISTORY**

65107 First released in Issue 1. Derived from Issue 1 of the SVID.

65108 **Issue 6**

65109 The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

65110 **Issue 7**

65111 The *stpcpy()* function is added from The Open Group Technical Standard, 2006, Extended API
65112 Set Part 1.

65113 **NAME**

65114 strcspn — get the length of a complementary substring

65115 **SYNOPSIS**

65116 #include <string.h>

65117 size_t strcspn(const char *s1, const char *s2);

65118 **DESCRIPTION**

65119 CX The functionality described on this reference page is aligned with the ISO C standard. Any
65120 conflict between the requirements described here and the ISO C standard is unintentional. This
65121 volume of POSIX.1-2017 defers to the ISO C standard.

65122 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the
65123 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

65124 **RETURN VALUE**

65125 The *strcspn()* function shall return the length of the computed segment of the string pointed to
65126 by *s1*; no return value is reserved to indicate an error.

65127 **ERRORS**

65128 No errors are defined.

65129 **EXAMPLES**

65130 None.

65131 **APPLICATION USAGE**

65132 None.

65133 **RATIONALE**

65134 None.

65135 **FUTURE DIRECTIONS**

65136 None.

65137 **SEE ALSO**65138 [strspn\(\)](#)65139 XBD [<string.h>](#)65140 **CHANGE HISTORY**

65141 First released in Issue 1. Derived from Issue 1 of the SVID.

65142 **Issue 5**

65143 The RETURN VALUE section is updated to indicate that *strcspn()* returns the length of *s1*, and
65144 not *s1* itself as was previously stated.

65145 **Issue 6**

65146 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is
65147 updated to indicate that the computed segment length is returned, not the *s1* length.

65148 **NAME**65149 *strdup*, *strndup* — duplicate a specific number of bytes from a string65150 **SYNOPSIS**

```
65151 CX      #include <string.h>
65152          char *strdup(const char *s);
65153          char *strndup(const char *s, size_t size);
```

65154 **DESCRIPTION**

65155 The *strdup*() function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s*. The returned pointer can be passed to *free*(). A null pointer is returned if the new string cannot be created.

65158 The *strndup*() function shall be equivalent to the *strdup*() function, duplicating the provided *s* in a new block of memory allocated as if by using *malloc*(), with the exception being that *strndup*() copies at most *size* plus one bytes into the newly allocated memory, terminating the new string with a NUL character. If the length of *s* is larger than *size*, only *size* bytes shall be duplicated. If *size* is larger than the length of *s*, all bytes in *s* shall be copied into the new memory buffer, including the terminating NUL character. The newly created string shall always be properly terminated.

65165 **RETURN VALUE**

65166 The *strdup*() function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

65168 Upon successful completion, the *strndup*() function shall return a pointer to the newly allocated memory containing the duplicated string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

65171 **ERRORS**

65172 These functions shall fail if:

65173 [ENOMEM] Storage space available is insufficient.

65174 **EXAMPLES**

65175 None.

65176 **APPLICATION USAGE**

65177 For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *strdup*() and *strndup*(), this is the return value.

65180 Implementations are free to *malloc*() a buffer containing either (*size* + 1) bytes or (*strlen*(*s*, *size*) + 1) bytes. Applications should not assume that *strndup*() will allocate (*size* + 1) bytes when *strlen*(*s*) is smaller than *size*.

65183 **RATIONALE**

65184 None.

65185 **FUTURE DIRECTIONS**

65186 None.

65187 **SEE ALSO**

65188 *free*(), *wcsdup*()

65189 XBD <[string.h](#)>

65190 CHANGE HISTORY

65191 First released in Issue 4, Version 2.

65192 Issue 5

65193 Moved from X/OPEN UNIX extension to BASE.

65194 Issue 7

65195 Austin Group Interpretation 1003.1-2001 #044 is applied, changing the ``may fail'' [ENOMEM]
65196 error to become a ``shall fail'' error.

65197 The *strdup()* function is moved from the XSI option to the Base.

65198 The *strndup()* function is added from The Open Group Technical Standard, 2006, Extended API
65199 Set Part 1.

65200 The APPLICATION USAGE section is updated to clarify that memory is allocated as if by
65201 *malloc()*.

65202 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0338 [738] is applied.

65203 **NAME**65204 `strerror, strerror_l, strerror_r` — get error message string65205 **SYNOPSIS**65206 `#include <string.h>`65207 `char *strerror(int errnum);`65208 CX `char *strerror_l(int errnum, locale_t locale);`65209 `int strerror_r(int errnum, char *strerrbuf, size_t buflen);`65210 **DESCRIPTION**

65211 CX For `strerror()`: The functionality described on this reference page is aligned with the ISO C
 65212 standard. Any conflict between the requirements described here and the ISO C standard is
 65213 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

65214 The `strerror()` function shall map the error number in `errnum` to a locale-dependent error
 65215 message string and shall return a pointer to it. Typically, the values for `errnum` come from `errno`,
 65216 but `strerror()` shall map any value of type `int` to a message.

65217 CX The application shall not modify the string returned. The returned string pointer might be
 65218 CX invalidated or the string content might be overwritten by a subsequent call to `strerror()`, or by a
 65219 subsequent call to `strerror_l()` in the same thread. The returned pointer and the string content
 65220 might also be invalidated if the calling thread is terminated.

65221 CX The string may be overwritten by a subsequent call to `strerror_l()` in the same thread.

65222 The contents of the error message strings returned by `strerror()` should be determined by the
 65223 setting of the `LC_MESSAGES` category in the current locale.

65224 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 65225 `strerror()`.

65226 CX The `strerror()` and `strerror_l()` functions shall not change the setting of `errno` if successful.

65227 Since no return value is reserved to indicate an error of `strerror()`, an application wishing to
 65228 check for error situations should set `errno` to 0, then call `strerror()`, then check `errno`. Similarly,
 65229 since `strerror_l()` is required to return a string for some errors, an application wishing to check
 65230 for all error situations should set `errno` to 0, then call `strerror_l()`, then check `errno`.

65231 The `strerror()` function need not be thread-safe.

65232 The `strerror_l()` function shall map the error number in `errnum` to a locale-dependent error
 65233 message string in the locale represented by `locale` and shall return a pointer to it.

65234 The `strerror_r()` function shall map the error number in `errnum` to a locale-dependent error
 65235 message string and shall return the string in the buffer pointed to by `strerrbuf`, with length
 65236 `buflen`.

65237 CX If the value of `errnum` is a valid error number, the message string shall indicate what error
 65238 occurred; if the value of `errnum` is zero, the message string shall either be an empty string or
 65239 indicate that no error occurred; otherwise, if these functions complete successfully, the message
 65240 string shall indicate that an unknown error occurred.

65241 CX The behavior is undefined if the `locale` argument to `strerror_l()` is the special locale object
 65242 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.

65243 **RETURN VALUE**

65244 Upon completion, whether successful or not, *strerror()* shall return a pointer to the generated
 65245 CX message string. On error *errno* may be set, but no return value is reserved to indicate an error.

65246 Upon successful completion, *strerror_l()* shall return a pointer to the generated message string. If
 65247 *errnum* is not a valid error number, *errno* may be set to [EINVAL], but a pointer to a message
 65248 string shall still be returned. If any other error occurs, *errno* shall be set to indicate the error and
 65249 a null pointer shall be returned.

65250 Upon successful completion, *strerror_r()* shall return 0. Otherwise, an error number shall be
 65251 returned to indicate the error.

65252 **ERRORS**

65253 These functions may fail if:

65254 CX [EINVAL] The value of *errnum* is neither a valid error number nor zero.

65255 The *strerror_r()* function may fail if:

65256 CX [ERANGE] Insufficient storage was supplied via *strrbuf* and *buflen* to contain the
 65257 generated message string.

65258 **EXAMPLES**

65259 None.

65260 **APPLICATION USAGE**

65261 Historically in some implementations, calls to *perror()* would overwrite the string that the
 65262 pointer returned by *strerror()* points to. Such implementations did not conform to the ISO C
 65263 standard; however, application developers should be aware of this behavior if they wish their
 65264 applications to be portable to such implementations.

65265 **RATIONALE**

65266 The *strerror_l()* function is required to be thread-safe, thereby eliminating the need for an
 65267 equivalent to the *strerror_r()* function.

65268 Earlier versions of this standard did not explicitly require that the error message strings returned
 65269 by *strerror()* and *strerror_r()* provide any information about the error. This version of the
 65270 standard requires a meaningful message for any successful completion.

65271 Since no return value is reserved to indicate a *strerror()* error, but all calls (whether successful or
 65272 not) must return a pointer to a message string, on error *strerror()* can return a pointer to an
 65273 empty string or a pointer to a meaningful string that can be printed.

65274 Note that the [EINVAL] error condition is a may fail error. If an invalid error number is supplied
 65275 as the value of *errnum*, applications should be prepared to handle any of the following:

- 65276 1. Error (with no meaningful message): *errno* is set to [EINVAL], the return value is a pointer
 65277 to an empty string.
- 65278 2. Successful completion: *errno* is unchanged and the return value points to a string like
 65279 "unknown error" or "error number xxx" (where xxx is the value of *errnum*).
- 65280 3. Combination of #1 and #2: *errno* is set to [EINVAL] and the return value points to a string
 65281 like "unknown error" or "error number xxx" (where xxx is the value of *errnum*).
 65282 Since applications frequently use the return value of *strerror()* as an argument to
 65283 functions like *fprintf()* (without checking the return value) and since applications have no
 65284 way to parse an error message string to determine whether *errnum* represents a valid
 65285 error number, implementations are encouraged to implement #3. Similarly,
 65286 implementations are encouraged to have *strerror_r()* return [EINVAL] and put a string

65287 like "unknown error" or "error number xxx" in the buffer pointed to by *strerrbuf*
 65288 when the value of *errno* is not a valid error number.

65289 Some applications rely on being able to set *errno* to 0 before calling a function with no reserved
 65290 value to indicate an error, then call *strerror(errno)* afterwards to detect whether an error occurred
 65291 (because *errno* changed) or to indicate success (because *errno* remained zero). This usage pattern
 65292 requires that *strerror(0)* succeed with useful results. Previous versions of the standard did not
 65293 specify the behavior when *errno* is zero.

65294 FUTURE DIRECTIONS

65295 None.

65296 SEE ALSO

65297 *perror()*

65298 XBD <[string.h](#)>

65299 CHANGE HISTORY

65300 First released in Issue 3.

65301 Issue 5

65302 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

65303 A note indicating that the *strerror()* function need not be reentrant is added to the
 65304 DESCRIPTION.

65305 Issue 6

65306 Extensions beyond the ISO C standard are marked.

65307 The following new requirements on POSIX implementations derive from alignment with the
 65308 Single UNIX Specification:

65309 In the RETURN VALUE section, the fact that *errno* may be set is added.

65310 The [EINVAL] optional error condition is added.

65311 The normative text is updated to avoid use of the term "must" for application requirements.

65312 The *strerror_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.

65313 The *strerror_r()* function is marked as part of the Thread-Safe Functions option.

65314 Issue 7

65315 Austin Group Interpretation 1003.1-2001 #072 is applied, updating the ERRORS section.

65316 Austin Group Interpretation 1003.1-2001 #156 is applied.

65317 Austin Group Interpretation 1003.1-2001 #187 is applied, clarifying the behavior when the
 65318 generated error message is an empty string.

65319 SD5-XSH-ERN-191 is applied, updating the APPLICATION USAGE section.

65320 The *strerror_l()* function is added from The Open Group Technical Standard, 2006, Extended API
 65321 Set Part 4.

65322 The *strerror_r()* function is moved from the Thread-Safe Functions option to the Base.

65323 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0595 [75], XSH/TC1-2008/0596 [447],
 65324 XSH/TC1-2008/0597 [382,428], XSH/TC1-2008/0598 [283], XSH/TC1-2008/0599 [382,428],
 65325 XSH/TC1-2008/0600 [283], and XSH/TC1-2008/0601 [382,428] are applied.



65326

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0339 [656] is applied.



65327 **NAME**

65328 strfmon, strfmon_l ¶convert monetary value to a string

65329 **SYNOPSIS**

65330 #include <monetary.h>

65331 ssize_t strfmon(char *restrict *s*, size_t *maxsize*,
65332 const char *restrict *format*, ...);65333 ssize_t strfmon_l(char *restrict *s*, size_t *maxsize*,
65334 locale_t *locale*, const char *restrict *format*, ...);65335 **DESCRIPTION**65336 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the
65337 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.65338 The format is a character string, beginning and ending in its initial state, if any, that contains two
65339 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*
65340 *specifications*, each of which shall result in the fetching of zero or more arguments which are
65341 converted and formatted. The results are undefined if there are insufficient arguments for the
65342 format. If the format is exhausted while arguments remain, the excess arguments are simply
65343 ignored.

65344 The application shall ensure that a conversion specification consists of the following sequence:

65345 A '%' character

65346 Optional flags

65347 Optional field width

65348 Optional left precision

65349 Optional right precision

65350 A required conversion specifier character that determines the conversion to be performed

65351 The *strfmon_l()* function shall be equivalent to the *strfmon()* function, except that the locale data
65352 used is from the locale represented by *locale*.65353 **Flags**

65354 One or more of the following optional flags can be specified to control the conversion:

65355 =*f* An '=' followed by a single character *f* which is used as the numeric fill character. In
65356 order to work with precision or width counts, the fill character shall be a single byte
65357 character; if not, the behavior is undefined. The default numeric fill character is the
65358 <space>. This flag does not affect field width filling which always uses the <space>.
65359 This flag is ignored unless a left precision (see below) is specified.65360 ^ Do not format the currency amount with grouping characters. The default is to insert
65361 the grouping characters if defined for the current locale.65362 + or (Specify the style of representing positive and negative currency amounts. Only one of
65363 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'
65364 are used (for example, in many locales, the empty string if positive and '-' if
65365 negative). If '(' is specified, negative amounts are enclosed within parentheses. If
65366 neither flag is specified, the '+' style is used.

65367 ! Suppress the currency symbol from the output conversion.

65368 – Specify the alignment. If this flag is present the result of the conversion is left-justified
 65369 (padded to the right) rather than right-justified. This flag shall be ignored unless a field
 65370 width (see below) is specified.

65371 Field Width

65372 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result
 65373 of the conversion is right-justified (or left-justified if the flag '-' is specified). The
 65374 default is 0.

65375 Left Precision

65376 *#n* A '#' followed by a decimal digit string *n* specifying a maximum number of digits
 65377 expected to be formatted to the left of the radix character. This option can be used to
 65378 keep the formatted output from multiple calls to the *strfmon()* function aligned in the
 65379 same columns. It can also be used to fill unused positions with a special character as in
 65380 "\$***123.45". This option causes an amount to be formatted as if it has the number
 65381 of digits specified by *n*. If more than *n* digit positions are required, this conversion
 65382 specification is ignored. Digit positions in excess of those actually required are filled
 65383 with the numeric fill character (see the *=f* flag above).

65384 If grouping has not been suppressed with the '^' flag, and it is defined for the current
 65385 locale, grouping separators are inserted before the fill characters (if any) are added.
 65386 Grouping separators are not applied to fill characters even if the fill character is a digit.

65387 To ensure alignment, any characters appearing before or after the number in the
 65388 formatted output such as currency or sign symbols are padded as necessary with
 65389 <space> characters to make their positive and negative formats an equal length.

65390 Right Precision

65391 *.p* A <period> followed by a decimal digit string *p* specifying the number of digits after
 65392 the radix character. If the value of the right precision *p* is 0, no radix character appears.
 65393 If a right precision is not included, a default specified by the current locale is used. The
 65394 amount being formatted is rounded to the specified number of digits prior to
 65395 formatting.

65396 Conversion Specifier Characters

65397 The conversion specifier characters and their meanings are:

65398 *i* The **double** argument is formatted according to the locale's international currency
 65399 format (for example, in the US: USD 1,234.56). If the argument is $\pm\text{Inf}$ or NaN, the result
 65400 of the conversion is unspecified.

65401 *n* The **double** argument is formatted according to the locale's national currency format
 65402 (for example, in the US: \$1,234.56). If the argument is $\pm\text{Inf}$ or NaN, the result of the
 65403 conversion is unspecified.

65404 *%* Convert to a '%'; no argument is converted. The entire conversion specification shall
 65405 be %%.

65406 **Locale Information**

65407 The *LC_MONETARY* category of the current locale affects the behavior of this function including
 65408 the monetary radix character (which may be different from the numeric radix character affected
 65409 by the *LC_NUMERIC* category), the grouping separator, the currency symbols, and formats. The
 65410 international currency symbol should be conformant with the ISO 4217:2001 standard.

65411 If the value of *maxsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

65412 The behavior is undefined if the *locale* argument to *strfmon_l()* is the special locale object
 65413 LC_GLOBAL_LOCALE or is not a valid locale object handle.

65414 **RETURN VALUE**

65415 If the total number of resulting bytes including the terminating null byte is not more than
 65416 *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*,
 65417 not including the terminating NUL character. Otherwise, -1 shall be returned, the contents of the
 65418 array are unspecified, and *errno* shall be set to indicate the error.

65419 **ERRORS**

65420 These functions shall fail if:

65421 [E2BIG] Conversion stopped due to lack of space in the buffer.

65422 **EXAMPLES**

65423 Given a locale for the US and the values 123.45, -123.45, and 3456.781, the following output
 65424 might be produced. Square brackets (" [] ") are used in this example to delimit the output.

65425	%n	[\$123.45]	Default formatting
65426		[-\$123.45]	
65427		[\$3,456.78]	
65428	%11n	[\$123.45]	Right align within an 11-character field
65429		[-\$123.45]	
65430		[\$3,456.78]	
65431	%#5n	[\$ 123.45]	Aligned columns for values up to 99999
65432		[-\$ 123.45]	
65433		[\$ 3,456.78]	
65434	%=*#5n	[\$***123.45]	Specify a fill character
65435		[-\$***123.45]	
65436		[\$*3,456.78]	
65437	%=0#5n	[\$000123.45]	Fill characters do not use grouping
65438		[-\$000123.45]	even if the fill character is a digit
65439		[\$03,456.78]	
65440	%^#5n	[\$ 123.45]	Disable the grouping separator
65441		[-\$ 123.45]	
65442		[\$ 3456.78]	
65443	%^#5.0n	[\$ 123]	Round off to whole units
65444		[-\$ 123]	
65445		[\$ 3457]	
65446	%^#5.4n	[\$ 123.4500]	Increase the precision
65447		[-\$ 123.4500]	
65448		[\$ 3456.7810]	
65449	%(#5n	[\$ 123.45]	Use an alternative pos/neg style


```

65450          [ ( $ 123.45 ) ]
65451          [ $ 3,456.78 ]

65452          %! ( #5n [ 123.45 ]      Disable the currency symbol
65453          [ ( 123.45 ) ]
65454          [ 3,456.78 ]

65455          %-14#5.4n [ $ 123.4500 ]  Left-justify the output
65456          [ -$ 123.4500 ]
65457          [ $ 3,456.7810 ]

65458          %14#5.4n [ $ 123.4500 ]  Corresponding right-justified output
65459          [ -$ 123.4500 ]
65460          [ $ 3,456.7810 ]

```

65461 See also the EXAMPLES section in *fprintf()*.

65462 APPLICATION USAGE

65463 None.

65464 RATIONALE

65465 None.

65466 FUTURE DIRECTIONS

65467 Lowercase conversion characters are reserved for future standards use and uppercase for
65468 implementation-defined use.

65469 SEE ALSO

65470 *fprintf()*, *localeconv()*

65471 XBD <[monetary.h](#)>

65472 CHANGE HISTORY

65473 First released in Issue 4.

65474 Issue 5

65475 Moved from ENHANCED I18N to BASE.

65476 The [ENOSYS] error is removed.

65477 Text is added to the DESCRIPTION warning about values of *maxsize* that are greater than
65478 {SSIZE_MAX}.

65479 Issue 6

65480 The normative text is updated to avoid use of the term “must” for application requirements.

65481 The **restrict** keyword is added to the *strfmon()* prototype for alignment with the
65482 ISO/IEC 9899:1999 standard.

65483 The EXAMPLES section is reworked, clarifying the output format.

65484 Issue 7

65485 SD5-XSH-ERN-29 is applied, updating the examples for % (#5n and %! (#5n.

65486 SD5-XSH-ERN-233 is applied, changing the definition of the '+' or '(' flags to refer to
65487 multiple locales.

65488 The *strfmon()* function is moved from the XSI option to the Base.

65489
65490

The *strfmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

65491
65492

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0602 [302], XSH/TC1-2008/0603 [283], and XSH/TC1-2008/0604 [283] are applied.

65493 **NAME**65494 `strptime, strptime_l` ¶convert date and time to a string65495 **SYNOPSIS**65496 `#include <time.h>`65497 `size_t strptime(char *restrict s, size_t maxsize,`
65498 `const char *restrict format, const struct tm *restrict timeptr);`65499 CX `size_t strptime_l(char *restrict s, size_t maxsize,`
65500 `const char *restrict format, const struct tm *restrict timeptr,`
65501 `locale_t locale);`65502 **DESCRIPTION**65503 CX For `strptime()`: The functionality described on this reference page is aligned with the ISO C
65504 standard. Any conflict between the requirements described here and the ISO C standard is
65505 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.65506 The `strptime()` function shall place bytes into the array pointed to by `s` as controlled by the string
65507 pointed to by `format`. The format is a character string, beginning and ending in its initial shift
65508 state, if any. The format string consists of zero or more conversion specifications and ordinary
65509 characters.65510 Each conversion specification is introduced by the '%' character after which the following
65511 appear in sequence:

65512 CX An optional flag:

65513 0 The zero character ('0'), which specifies that the character used as the padding
65514 character is '0',65515 + The <plus-sign> character ('+'), which specifies that the character used as the
65516 padding character is '0', and that if and only if the field being produced consumes
65517 more than four bytes to represent a year (for %F, %G, or %Y) or more than two bytes to
65518 represent the year divided by 100 (for %C) then a leading <plus-sign> character shall
65519 be included if the year being processed is greater than or equal to zero or a leading
65520 <hyphen-minus> character ('-') shall be included if the year is less than zero.

65521 The default padding character is unspecified.

65522 An optional minimum field width. If the converted value, including any leading '+' or
65523 '-' sign, has fewer bytes than the minimum field width and the padding character is not
65524 the NUL character, the output shall be padded on the left (after any leading '+' or '-'
65525 sign) with the padding character.

65526 An optional E or O modifier.

65527 A terminating conversion specifier character that indicates the type of conversion to be
65528 applied.65529 CX The results are unspecified if more than one flag character is specified, a flag character is
65530 specified without a minimum field width; a minimum field width is specified without a flag
65531 character; a modifier is specified with a flag or with a minimum field width; or if a minimum
65532 field width is specified for any conversion specifier other than C, F, G, or Y.65533 All ordinary characters (including the terminating NUL character) are copied unchanged into
65534 the array. If copying takes place between objects that overlap, the behavior is undefined. No
65535 more than `maxsize` bytes are placed into the array. Each conversion specifier is replaced by
65536 appropriate characters as described in the following list. The appropriate characters are

- 65537 determined using the *LC_TIME* category of the current locale and by the values of zero or more
 65538 members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the
 65539 description. If any of the specified values are outside the normal range, the characters stored are
 65540 unspecified.
- 65541 CX The *strptime_l()* function shall be equivalent to the *strptime()* function, except that the locale data
 65542 used is from the locale represented by *locale*.
- 65543 Local timezone information is used as though *strptime()* called *tzset()*.
- 65544 The following conversion specifiers shall be supported:
- 65545 a Replaced by the locale's abbreviated weekday name. [*tm_wday*]
- 65546 A Replaced by the locale's full weekday name. [*tm_wday*]
- 65547 b Replaced by the locale's abbreviated month name. [*tm_mon*]
- 65548 B Replaced by the locale's full month name. [*tm_mon*]
- 65549 c Replaced by the locale's appropriate date and time representation. (See the Base
 65550 Definitions volume of POSIX.1-2017, <**time.h**>.)
- 65551 C Replaced by the year divided by 100 and truncated to an integer, as a decimal number.
 65552 [*tm_year*]
- 65553 If a minimum field width is not specified, the number of characters placed into the
 65554 array pointed to by *s* will be the number of digits in the year divided by 100 or two,
 65555 CX whichever is greater. If a minimum field width is specified, the number of characters
 65556 placed into the array pointed to by *s* will be the number of digits in the year divided by
 65557 100 or the minimum field width, whichever is greater.
- 65558 d Replaced by the day of the month as a decimal number [01,31]. [*tm_mday*]
- 65559 D Equivalent to *%m/%d/%y*. [*tm_mon, tm_mday, tm_year*]
- 65560 e Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded
 65561 by a space. [*tm_mday*]
- 65562 CX F Equivalent to *%+4Y-%m-%d* if no flag and no minimum field width are specified.
 65563 [*tm_year, tm_mon, tm_mday*]
- 65564 CX If a minimum field width of *x* is specified, the year shall be output as if by the *Y*
 65565 specifier (described below) with whatever flag was given and a minimum field width
 65566 of *x*-6. If *x* is less than 6, the behavior shall be as if *x* equalled 6.
- 65567 If the minimum field width is specified to be 10, and the year is four digits long, then
 65568 the output string produced will match the ISO 8601:2004 standard subclause 4.1.2.2
 65569 complete representation, extended format date representation of a specific day. If a +
 65570 flag is specified, a minimum field width of *x* is specified, and *x*-7 bytes are sufficient to
 65571 hold the digits of the year (not including any needed sign character), then the output
 65572 will match the ISO 8601:2004 standard subclause 4.1.2.4 complete representation,
 65573 expanded format date representation of a specific day.
- 65574 g Replaced by the last 2 digits of the week-based year (see below) as a decimal number
 65575 [00,99]. [*tm_year, tm_wday, tm_yday*]
- 65576 G Replaced by the week-based year (see below) as a decimal number (for example, 1977).
 65577 [*tm_year, tm_wday, tm_yday*]

65578	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> will be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
65579		
65580		
65581	h	Equivalent to %b. [<i>tm_mon</i>]
65582	H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [<i>tm_hour</i>]
65583	I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [<i>tm_hour</i>]
65584	j	Replaced by the day of the year as a decimal number [001,366]. [<i>tm_yday</i>]
65585	m	Replaced by the month as a decimal number [01,12]. [<i>tm_mon</i>]
65586	M	Replaced by the minute as a decimal number [00,59]. [<i>tm_min</i>]
65587	n	Replaced by a <newline>.
65588	p	Replaced by the locale's equivalent of either a.m. or p.m. [<i>tm_hour</i>]
65589	CX	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be equivalent to %I:%M:%S %p. [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
65590		
65591	R	Replaced by the time in 24-hour notation (%H:%M). [<i>tm_hour</i> , <i>tm_min</i>]
65592	S	Replaced by the second as a decimal number [00,60]. [<i>tm_sec</i>]
65593	t	Replaced by a <tab>.
65594	T	Replaced by the time (%H:%M:%S). [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
65595	u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [<i>tm_wday</i>]
65596		
65597	U	Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
65598		
65599		
65600	V	Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
65601		
65602		
65603		
65604		
65605	w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [<i>tm_wday</i>]
65606		
65607	W	Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
65608		
65609		
65610	x	Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-2017, <time.h>.)
65611		
65612	X	Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-2017, <time.h>.)
65613		
65614	y	Replaced by the last two digits of the year as a decimal number [00,99]. [<i>tm_year</i>]
65615	Y	Replaced by the year as a decimal number (for example, 1997). [<i>tm_year</i>]
65616	CX	If a minimum field width is specified, the number of characters placed into the array pointed to by <i>s</i> will be the number of digits and leading sign characters (if any) in the
65617		

65618		year, or the minimum field width, whichever is greater.
65619	z	Replaced by the offset from UTC in the ISO 8601:2004 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable. For example, "-0430" means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [<i>tm_isdst</i>]
65620		
65621	CX	
65622		
65623		
65624	Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [<i>tm_isdst</i>]
65625		
65626	%	Replaced by %.
65627		If a conversion specification does not correspond to any of the above, the behavior is undefined.
65628	CX	If a struct tm broken-down time structure is created by <i>localtime()</i> or <i>localtime_r()</i> , or modified by <i>mktime()</i> , and the value of <i>TZ</i> is subsequently modified, the results of the %Z and %z <i>strptime()</i> conversion specifiers are undefined, when <i>strptime()</i> is called with such a broken-down time structure.
65629		
65630		
65631		
65632		If a struct tm broken-down time structure is created or modified by <i>gmtime()</i> or <i>gmtime_r()</i> , it is unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the current local timezone, when <i>strptime()</i> is called with such a broken-down time structure.
65633		
65634		
65635		Modified Conversion Specifiers
65636		Some conversion specifiers can be modified by the E or O modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in XBD Section 7.3.5, on page 159), the behavior shall be as if the unmodified conversion specification were used.
65637		
65638		
65639		
65640		
65641	%Ec	Replaced by the locale's alternative appropriate date and time representation.
65642	%EC	Replaced by the name of the base year (period) in the locale's alternative representation.
65643		
65644	%Ex	Replaced by the locale's alternative date representation.
65645	%EX	Replaced by the locale's alternative time representation.
65646	%Ey	Replaced by the offset from %EC (year only) in the locale's alternative representation.
65647	%EY	Replaced by the full alternative year representation.
65648	%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading <space> characters.
65649		
65650		
65651	%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading <space> characters.
65652		
65653	%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.
65654	%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.
65655	%Om	Replaced by the month using the locale's alternative numeric symbols.
65656	%OM	Replaced by the minutes using the locale's alternative numeric symbols.

65657	%OS	Replaced by the seconds using the locale's alternative numeric symbols.
65658	%Ou	Replaced by the weekday as a number in the locale's alternative representation (Monday=1).
65659		
65660	%OU	Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to %U) using the locale's alternative numeric symbols.
65661		
65662	%OV	Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to %V) using the locale's alternative numeric symbols.
65663		
65664	%Ow	Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
65665		
65666	%OW	Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
65667		
65668	%Oy	Replaced by the year (offset from %C) using the locale's alternative numeric symbols.
65669		
65670		%g, %G, and %V give values according to the ISO 8601:2004 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, %G is replaced by 1998 and %V is replaced by 01.
65671		
65672		
65673		
65674		
65675		
65676		
65677		If a conversion specifier is not one of the above, the behavior is undefined.
65678	CX	The behavior is undefined if the <i>locale</i> argument to <i>strptime_l()</i> is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.
65679		

65680 RETURN VALUE

65681 If the total number of resulting bytes including the terminating null byte is not more than
 65682 *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*,
 65683 not including the terminating NUL character. Otherwise, 0 shall be returned and the contents of
 65684 the array are unspecified.

65685 ERRORS

65686 No errors are defined.

65687 EXAMPLES

65688 Getting a Localized Date String

65689 The following example first sets the locale to the user's default. The locale information will be
 65690 used in the *nl_langinfo()* and *strptime()* functions. The *nl_langinfo()* function returns the localized
 65691 date string which specifies how the date is laid out. The *strptime()* function takes this
 65692 information and, using the **tm** structure for values, places the date and time information into
 65693 *datestring*.

```
65694 #include <time.h>
65695 #include <locale.h>
65696 #include <langinfo.h>
65697 ...
65698 struct tm *tm;
65699 char datestring[256];
65700 ...
```

```

65701     setlocale (LC_ALL, "");
65702     ...
65703     strptime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
65704     ...

```

APPLICATION USAGE

65705 The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

65706 Some of the conversion specifications are duplicates of others. They are included for compatibility with *nl_cxtime()* and *nl_ascxtime()*, which were published in Issue 2.

65707 The %C, %F, %G, and %Y format specifiers in *strptime()* always print full values, but the *strptime()*
65708 %C, %F, and %Y format specifiers only scan two digits (assumed to be the first two digits of a
65709 four-digit year) for %C and four digits (assumed to be the entire (four-digit) year) for %F and %Y.
65710 This mimics the behavior of *printf()* and *scanf()*; that is:

```
65711 printf("%2d", x = 1000);
```

65712 prints "1000", but:

```
65713 scanf("%2d", &x);
```

65714 when given "1000" as input will only store 10 in *x*). Applications using extended ranges of
65715 years must be sure that the number of digits specified for scanning years with *strptime()* matches
65716 the number of digits that will actually be present in the input stream. Historic implementations
65717 of the %Y conversion specification (with no flags and no minimum field width) produced
65718 different output formats. Some always produced at least four digits (with 0 fill for years from 0
65719 through 999) while others only produced the number of digits present in the year (with no fill
65720 and no padding). These two forms can be produced with the '0' flag and a minimum field
65721 width options using the conversions specifications %04Y and %01Y, respectively.

65722 In the past, the C and POSIX standards specified that %F produced an ISO 8601:2004 standard
65723 date format, but didn't specify which one. For years in the range [0001,9999], POSIX.1-2017
65724 requires that the output produced match the ISO 8601:2004 standard complete representation
65725 extended format (YYYY-MM-DD) and for years outside of this range produce output that
65726 matches the ISO 8601:2004 standard expanded representation extended format
65727 (<+/-><Underline>Y</Underline>YYYY-MM-DD). To fully meet ISO 8601:2004 standard
65728 requirements, the producer and consumer must agree on a date format that has a specific
65729 number of bytes reserved to hold the characters used to represent the years that is sufficiently
65730 large to hold all values that will be shared. For example, the %+13F conversion specification will
65731 produce output matching the format "<+/->YYYYYY-MM-DD" (a leading '+' or '-' sign; a six-
65732 digit, 0-filled year; a '-' ; a two-digit, leading 0-filled month; another '-' ; and the two-digit,
65733 leading 0-filled day within the month).

65734 Note that if the year being printed is greater than 9999, the resulting string from the unadorned
65735 %F conversion specifications will not conform to the ISO 8601:2004 standard extended format,
65736 complete representation for a date and will instead be an extended format, expanded
65737 representation (presumably without the required agreement between the date's producer and
65738 consumer).

65739 In the C or POSIX locale, the E and O modifiers are ignored and the replacement strings for the
65740 following specifiers are:

65741 %a The first three characters of %A.

65742 %A One of Sunday, Monday, ..., Saturday.

65745	%b	The first three characters of %B.
65746	%B	One of January, February, . . . , December.
65747	%c	Equivalent to %a %b %e %T %Y.
65748	%p	One of AM or PM.
65749	%r	Equivalent to %I:%M:%S %p.
65750	%x	Equivalent to %m/%d/%y.
65751	%X	Equivalent to %T.
65752	%Z	Implementation-defined.

65753 RATIONALE

65754 The %Y conversion specification to *strptime()* was frequently assumed to be a four-digit year, but
 65755 the ISO C standard does not specify that %Y is restricted to any subset of allowed values from the
 65756 *tm_year* field. Similarly, the %C conversion specification was assumed to be a two-digit field and
 65757 the first part of the output from the %F conversion specification was assumed to be a four-digit
 65758 field. With *tm_year* being a signed 32 or more-bit **int** and with many current implementations
 65759 supporting 64-bit **time_t** types in one or more programming environments, these assumptions
 65760 are clearly wrong.

65761 POSIX.1-2017 now allows the format specifications %0xC, %0xF, %0xG, and %0xY (where 'x' is
 65762 a string of decimal digits used to specify printing and scanning of a string of *x* decimal digits)
 65763 with leading zero fill characters. Allowing applications to set the field width enables them to
 65764 agree on the number of digits to be printed and scanned in the ISO 8601:2004 standard
 65765 expanded representation of a year (for %F, %G, and %Y) or all but the last two digits of the year
 65766 (for %C). This is based on a feature in some versions of GNU **libc**'s *strptime()*. The GNU version
 65767 allows specifying space, zero, or no-fill characters in *strptime()* format strings, but does not allow
 65768 any flags to be specified in *strptime()* format strings. These implementations also allow these
 65769 flags to be specified for any numeric field. POSIX.1-2017 only requires the zero fill flag ('0') and
 65770 only requires that it be recognized when processing %C, %F, %G, and %Y specifications when a
 65771 minimum field width is also specified. The '0' flag is the only flag needed to produce and scan
 65772 the ISO 8601:2004 standard year fields using the extended format forms. POSIX.1-2017 also
 65773 allows applications to specify the same flag and field width specifiers to be used in both
 65774 *strptime()* and *strptime()* format strings for symmetry. Systems may provide other flag characters
 65775 and may accept flags in conjunction with conversion specifiers other than %C, %F, %G, and %Y;
 65776 but portable applications cannot depend on such extensions.

65777 POSIX.1-2017 now also allows the format specifications %+xC, %+xF, %+xG, and %+xY (where
 65778 'x' is a string of decimal digits used to specify printing and scanning of a string of 'x' decimal
 65779 digits) with leading zero fill characters and a leading '+' sign character if the year being
 65780 converted is more than four digits or a minimum field width is specified that allows room for
 65781 more than four digits for the year. This allows date providers and consumers to agree on a
 65782 specific number of digits to represent a year as required by the ISO 8601:2004 standard
 65783 expanded representation formats. The expanded representation formats all require the year to
 65784 begin with a leading '+' or '-' sign. (All of these specifiers can also provide a leading '-'
 65785 sign for negative years. Since negative years and the year 0 don't fit well with the Gregorian or
 65786 Julian calendars, the normal ranges of dates start with year 1. The ISO C standard allows *tm_year*
 65787 to assume values corresponding to years before year 1, but the use of such years provided
 65788 unspecified results.)

65789 Some earlier version of this standard specified that applications wanting to use *strptime()* to scan
 65790 dates and times printed by *strptime()* should provide non-digit characters between fields to

65791 separate years from months and days. It also supported %F to print and scan the ISO 8601:2004
 65792 standard extended format, complete representation date for years 1 through 9999 (i.e., YYYY-
 65793 MM-DD). However, many applications were written to print (using *strptime()*) and scan (using
 65794 *strptime()*) dates written using the basic format complete representation (four-digit years) and
 65795 truncated representation (two-digit years) specified by the ISO 8601:2004 standard
 65796 representation of dates and times which do not have any separation characters between fields.
 65797 The ISO 8601:2004 standard also specifies basic format expanded representation where the
 65798 creator and consumer of these fields agree beforehand to represent years as leading zero-filled
 65799 strings of an agreed length of more than four digits to represent a year (again with no separation
 65800 characters when year, month, and day are all displayed). Applications producing and
 65801 consuming expanded representations are encouraged to use the '+' flag and an appropriate
 65802 maximum field width to scan the year including the leading sign. Note that even without the
 65803 '+' flag, years less than zero may be represented with a leading <hyphen-minus> for %F, %G,
 65804 and %Y conversion specifications. Using negative years results in unspecified behavior.

65805 If a format specification %*x*F with the field width *x* greater than 11 is specified and the width is
 65806 large enough to display the full year, the output string produced will match the ISO 8601:2004
 65807 standard subclause 4.1.2.4 expanded representation, extended format date representation for a
 65808 specific day. (For years in the range [1,99999], %+12F is sufficient for an agreed five-digit year
 65809 with a leading sign using the ISO 8601:2004 standard expanded representation, extended format
 65810 for a specific day "<+/->YYYY-MM-DD".) Note also that years less than 0 may produce a
 65811 leading <hyphen-minus> character ('-') when using %Y or %C whether or not the '0' or '+'
 65812 flags are used.

65813 The difference between the '0' flag and the '+' flag is whether the leading '+' character will
 65814 be provided for years >9999 as required for the ISO 8601:2004 standard extended representation
 65815 format containing a year. For example:

Year	Conversion Specification	<i>strptime()</i> Output	<i>strptime()</i> Scan Back
1970	%Y	1970	1970
1970	%+4Y	1970	1970
27	%Y	27 or 0027	27
270	%Y	270 or 0270	270
270	%+4Y	0270	270
17	%C%y	0017	17
270	%C%y	0270	270
12345	%Y	12345	1234*
12345	%+4Y	+12345	123*
12345	%05Y	12345	12345
270	%+5Y or %+3C%y	+0270	270
12345	%+5Y or %+3C%y	+12345	1234*
12345	%06Y or %04C%y	012345	12345
12345	%+6Y or %+4C%y	+12345	12345
123456	%08Y or %06C%y	00123456	123456
123456	%+8Y or %+6C%y	+0123456	123456

65834 In the cases above marked with a * in the *strptime()* scan back field, the implied or specified
 65835 number of characters scanned by *strptime()* was less than the number of characters output by
 65836 *strptime()* using the same format; so the remaining digits of the year were dropped when the
 65837 output date produced by *strptime()* was scanned back in by *strptime()*.

65838 **FUTURE DIRECTIONS**

65839 None.

65840 **SEE ALSO**65841 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *time()*,
65842 *tzset()*, *uselocale()*, *utime()*65843 XBD [Section 7.3.5](#) (on page 159), [<time.h>](#)65844 **CHANGE HISTORY**

65845 First released in Issue 3.

65846 **Issue 5**65847 The description of %OV is changed to be consistent with %V and defines Monday as the first day
65848 of the week.

65849 The description of %Oy is clarified.

65850 **Issue 6**

65851 Extensions beyond the ISO C standard are marked.

65852 The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from
65853 ``Otherwise, it is week 53 of the previous year, and the next week is week 1'' to ``Otherwise, it is
65854 the last week of the previous year, and the next week is week 1''.65855 The following new requirements on POSIX implementations derive from alignment with the
65856 Single UNIX Specification:

65857 The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.

65858 The modified conversion specifiers are added for consistency with the ISO POSIX-2
65859 standard *date* utility.

65860 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

65861 The *strptime()* prototype is updated.

65862 The DESCRIPTION is extensively revised.

65863 The %z conversion specifier is added.

65864 An example is added.

65865 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

65866 **Issue 7**

65867 Austin Group Interpretation 1003.1-2001 #163 is applied.

65868 The *strptime_1()* function is added from The Open Group Technical Standard, 2006, Extended API
65869 Set Part 4.65870 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0605 [283], XSH/TC1-2008/0606 [283],
65871 XSH/TC1-2008/0607 [193], and XSH/TC1-2008/0608 [193] are applied.65872 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0340 [584], XSH/TC2-2008/0341 [796],
65873 XSH/TC2-2008/0342 [584], and XSH/TC2-2008/0343 [584] are applied.

65874 **NAME**

65875 strlen, strlen ‡get length of fixed size string

65876 **SYNOPSIS**

65877 #include <string.h>

65878 size_t strlen(const char *s);

65879 CX size_t strnlen(const char *s, size_t maxlen);

65880 **DESCRIPTION**

65881 CX For *strlen()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

65884 The *strlen()* function shall compute the number of bytes in the string to which *s* points, not including the terminating NUL character.

65886 CX The *strnlen()* function shall compute the smaller of the number of bytes in the array to which *s* points, not including any terminating NUL character, or the value of the *maxlen* argument. The *strnlen()* function shall never examine more than *maxlen* bytes of the array pointed to by *s*.

65889 **RETURN VALUE**

65890 The *strlen()* function shall return the length of *s*; no return value shall be reserved to indicate an error.

65892 CX The *strnlen()* function shall return the number of bytes preceding the first null byte in the array to which *s* points, if *s* contains a null byte within the first *maxlen* bytes; otherwise, it shall return *maxlen*.

65895 **ERRORS**

65896 No errors are defined.

65897 **EXAMPLES**65898 **Getting String Lengths**

65899 The following example sets the maximum length of *key* and *data* by using *strlen()* to get the lengths of those strings.

```
65901         #include <string.h>
65902         ...
65903         struct element {
65904             char *key;
65905             char *data;
65906         };
65907         ...
65908         char *key, *data;
65909         int len;

65910         *keylength = *datalength = 0;
65911         ...
65912         if ((len = strlen(key)) > *keylength)
65913             *keylength = len;
65914         if ((len = strlen(data)) > *datalength)
65915             *datalength = len;
65916         ...
```

65917 APPLICATION USAGE

65918 None.

65919 RATIONALE

65920 None.

65921 FUTURE DIRECTIONS

65922 None.

65923 SEE ALSO

65924 [wcslen\(\)](#)

65925 XBD [<string.h>](#)

65926 CHANGE HISTORY

65927 First released in Issue 1. Derived from Issue 1 of the SVID.

65928 Issue 5

65929 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not
65930 *s* itself as was previously stated.

65931 Issue 7

65932 The *strnlen()* function is added from The Open Group Technical Standard, 2006, Extended API
65933 Set Part 1.

65934 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0344 [560] is applied.

65935 **NAME**

65936 strncasecmp, strncasecmp_l ‡'case-insensitive string comparisons

65937 **SYNOPSIS**

65938 #include <strings.h>

65939 int strncasecmp(const char *s1, const char *s2, size_t n);

65940 int strncasecmp_l(const char *s1, const char *s2,

65941 size_t n, locale_t locale);

65942 **DESCRIPTION**65943 Refer to *strcasecmp()*.

65944 **NAME**

65945 strncat ‡concatenate a string with part of another

65946 **SYNOPSIS**

65947 #include <string.h>

65948 char *strncat(char *restrict s1, const char *restrict s2, size_t n);

65949 **DESCRIPTION**

65950 CX The functionality described on this reference page is aligned with the ISO C standard. Any
65951 conflict between the requirements described here and the ISO C standard is unintentional. This
65952 volume of POSIX.1-2017 defers to the ISO C standard.

65953 The *strncat*(*n*) function shall append not more than *n* bytes (a NUL character and bytes that
65954 follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by
65955 *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL
65956 character is always appended to the result. If copying takes place between objects that overlap,
65957 the behavior is undefined.

65958 **RETURN VALUE**65959 The *strncat*(*n*) function shall return *s1*; no return value shall be reserved to indicate an error.65960 **ERRORS**

65961 No errors are defined.

65962 **EXAMPLES**

65963 None.

65964 **APPLICATION USAGE**

65965 None.

65966 **RATIONALE**

65967 None.

65968 **FUTURE DIRECTIONS**

65969 None.

65970 **SEE ALSO**65971 [strcat\(\)](#)65972 XBD [<string.h>](#)65973 **CHANGE HISTORY**

65974 First released in Issue 1. Derived from Issue 1 of the SVID.

65975 **Issue 6**65976 The *strncat*(*n*) prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

65977 **NAME**

65978 strncmp ǂ'compare part of two strings

65979 **SYNOPSIS**

65980 #include <string.h>

65981 int strncmp(const char *s1, const char *s2, size_t n);

65982 **DESCRIPTION**

65983 CX The functionality described on this reference page is aligned with the ISO C standard. Any
65984 conflict between the requirements described here and the ISO C standard is unintentional. This
65985 volume of POSIX.1-2017 defers to the ISO C standard.

65986 The *strncmp()* function shall compare not more than *n* bytes (bytes that follow a NUL character
65987 are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

65988 The sign of a non-zero return value is determined by the sign of the difference between the
65989 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
65990 being compared.

65991 **RETURN VALUE**

65992 Upon successful completion, *strncmp()* shall return an integer greater than, equal to, or less than
65993 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the
65994 possibly null-terminated array pointed to by *s2* respectively.

65995 **ERRORS**

65996 No errors are defined.

65997 **EXAMPLES**

65998 None.

65999 **APPLICATION USAGE**

66000 None.

66001 **RATIONALE**

66002 None.

66003 **FUTURE DIRECTIONS**

66004 None.

66005 **SEE ALSO**66006 [strcmp\(\)](#)66007 XBD [<string.h>](#)66008 **CHANGE HISTORY**

66009 First released in Issue 1. Derived from Issue 1 of the SVID.

66010 **Issue 6**

66011 Extensions beyond the ISO C standard are marked.

66012 **NAME**

66013 stpnncpy, strncpy — copy fixed length string, returning a pointer to the array end

66014 **SYNOPSIS**

66015 #include <string.h>

66016 CX char *stpnncpy(char *restrict s1, const char *restrict s2, size_t n);

66017 char *strncpy(char *restrict s1, const char *restrict s2, size_t n);

66018 **DESCRIPTION**66019 CX For *strncpy()*: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.66022 CX The *stpnncpy()* and *strncpy()* functions shall copy not more than *n* bytes (bytes that follow a NUL character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.66024 If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

66026 If copying takes place between objects that overlap, the behavior is undefined.

66027 **RETURN VALUE**66028 CX If a NUL character is written to the destination, the *stpnncpy()* function shall return the address of the first such NUL character. Otherwise, it shall return *&s1[n]*.66030 The *strncpy()* function shall return *s1*.

66031 No return values are reserved to indicate an error.

66032 **ERRORS**

66033 No errors are defined.

66034 **EXAMPLES**

66035 None.

66036 **APPLICATION USAGE**66037 Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that the *s2* and *s1* arrays do not overlap.

66039 Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

66041 If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-terminated.66043 **RATIONALE**

66044 None.

66045 **FUTURE DIRECTIONS**

66046 None.

66047 **SEE ALSO**66048 *strcpy()*, *wcsncpy()*

66049 XBD <string.h>

66050 **CHANGE HISTORY**

66051 First released in Issue 1. Derived from Issue 1 of the SVID.

66052 **Issue 6**

66053 The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899: 1999 standard.

66054 **Issue 7**

66055 The *stpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API
66056 Set Part 1.

66057 **NAME**

66058 strndup — duplicate a specific number of bytes from a string

66059 **SYNOPSIS**

66060 CX #include <string.h>

66061 char *strndup(const char *s, size_t size);

66062 **DESCRIPTION**

66063 Refer to *strdup()*.

66064 **NAME**

66065 strnlen ‡'get length of fixed size string

66066 **SYNOPSIS**

66067 CX #include <string.h>

66068 size_t strnlen(const char *s, size_t maxlen);

66069 **DESCRIPTION**66070 Refer to *strlen()*.

66071 **NAME**

66072 strpbrk ‡scan a string for a byte

66073 **SYNOPSIS**

66074 #include <string.h>

66075 char *strpbrk(const char *s1, const char *s2);

66076 **DESCRIPTION**

66077 CX The functionality described on this reference page is aligned with the ISO C standard. Any
66078 conflict between the requirements described here and the ISO C standard is unintentional. This
66079 volume of POSIX.1-2017 defers to the ISO C standard.

66080 The *strpbrk()* function shall locate the first occurrence in the string pointed to by *s1* of any byte
66081 from the string pointed to by *s2*.

66082 **RETURN VALUE**

66083 Upon successful completion, *strpbrk()* shall return a pointer to the byte or a null pointer if no
66084 byte from *s2* occurs in *s1*.

66085 **ERRORS**

66086 No errors are defined.

66087 **EXAMPLES**

66088 None.

66089 **APPLICATION USAGE**

66090 None.

66091 **RATIONALE**

66092 None.

66093 **FUTURE DIRECTIONS**

66094 None.

66095 **SEE ALSO**66096 [strchr\(\)](#), [strrchr\(\)](#)66097 XBD [<string.h>](#)66098 **CHANGE HISTORY**

66099 First released in Issue 1. Derived from Issue 1 of the SVID.

66100 **NAME**

66101 strptime ¶date and time conversion

66102 **SYNOPSIS**

```
66103 XSI #include <time.h>
66104 char *strptime(const char *restrict buf, const char *restrict format,
66105               struct tm *restrict tm);
```

66106 **DESCRIPTION**

66107 The *strptime()* function shall convert the character string pointed to by *buf* to values which are
 66108 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

66109 The format is composed of zero or more directives. Each directive is composed of one of the
 66110 following: one or more white-space characters (as specified by *isspace()*); an ordinary character
 66111 (neither '%' nor a white-space character); or a conversion specification.

66112 Each conversion specification is introduced by the '%' character after which the following
 66113 appear in sequence:

66114 An optional flag, the zero character ('0') or the <plus-sign> character ('+'), which is
 66115 ignored.

66116 An optional field width. If a field width is specified, it shall be interpreted as a string of
 66117 decimal digits that will determine the maximum number of bytes converted for the
 66118 conversion rather than the number of bytes specified below in the description of the
 66119 conversion specifiers.

66120 An optional E or O modifier.

66121 A terminating conversion specifier character that indicates the type of conversion to be
 66122 applied.

66123 The conversions are determined using the *LC_TIME* category of the current locale. The
 66124 application shall ensure that there is white-space or other non-alphanumeric characters between
 66125 any two conversion specifications unless all of the adjacent conversion specifications convert a
 66126 known, fixed number of characters. In the following list, the maximum number of characters
 66127 scanned (excluding the one matching the next directive) is as follows:

66128 If a maximum field width is specified, then that number

66129 Otherwise, the pattern "{x}" indicates that the maximum is *x*

66130 Otherwise, the pattern "[x,y]" indicates that the value shall fall within the range given
 66131 (both bounds being inclusive), and the maximum number of characters scanned shall be
 66132 the maximum required to represent any value in the range without leading zeros and
 66133 without a leading <plus-sign>

66134 The following conversion specifiers are supported.

66135 The results are unspecified if a modifier is specified with a flag or with a minimum field width,
 66136 or if a field width is specified for any conversion specifier other than C or Y.

66137 a The day of the week, using the locale's weekday names; either the abbreviated or full
 66138 name may be specified.

66139 A Equivalent to %a.

66140	b	The month, using the locale's month names; either the abbreviated or full name may be specified.
66141		
66142	B	Equivalent to %b.
66143	c	Replaced by the locale's appropriate date and time representation.
66144	C	All but the last two digits of the year {2}; leading zeros shall be permitted but shall not be required. A leading '+' or '-' character shall be permitted before any leading zeros but shall not be required.
66145		
66146		
66147	d	The day of the month [01,31]; leading zeros shall be permitted but shall not be required.
66148	D	The date as %m/%d/%y.
66149	e	Equivalent to %d.
66150	h	Equivalent to %b.
66151	H	The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be required.
66152		
66153	I	The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be required.
66154		
66155	j	The day number of the year [001,366]; leading zeros shall be permitted but shall not be required.
66156		
66157	m	The month number [01,12]; leading zeros shall be permitted but shall not be required.
66158	M	The minute [00,59]; leading zeros shall be permitted but shall not be required.
66159	n	Any white space.
66160	p	The locale's equivalent of a.m. or p.m.
66161	r	12-hour clock time using the AM/PM notation if <code>t_fmt_ampm</code> is not an empty string in the <code>LC_TIME</code> portion of the current locale; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
66162		
66163		
66164	R	The time as %H:%M.
66165	S	The seconds [00,60]; leading zeros shall be permitted but shall not be required.
66166	t	Any white space.
66167	T	The time as %H:%M:%S.
66168	U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
66169		
66170	w	The weekday as a decimal number [0,6], with 0 representing Sunday.
66171	W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
66172		
66173	x	The date, using the locale's date format.
66174	X	The time, using the locale's time format.
66175	y	The last two digits of the year. When <i>format</i> contains neither a C conversion specifier nor a Y conversion specifier, values in the range [69,99] shall refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required. A leading '+' or '-'
66176		
66177		
66178		

- 66179 character shall be permitted before any leading zeros but shall not be required.
- 66180 **Note:** It is expected that in a future version of this standard the default century inferred
66181 from a 2-digit year will change. (This would apply to all commands accepting a
66182 2-digit year as input.)
- 66183 Y The full year {4}; leading zeros shall be permitted but shall not be required. A leading
66184 '+' or '-' character shall be permitted before any leading zeros but shall not be
66185 required.
- 66186 % Replaced by %.

66187 Modified Conversion Specifiers

66188 Some conversion specifiers can be modified by the E and O modifier characters to indicate that
66189 an alternative format or specification should be used rather than the one normally used by the
66190 unmodified conversion specifier. If the alternative format or specification does not exist in the
66191 current locale, the behavior shall be as if the unmodified conversion specification were used.

- 66192 %Ec The locale's alternative appropriate date and time representation.
- 66193 %EC The name of the base year (period) in the locale's alternative representation.
- 66194 %Ex The locale's alternative date representation.
- 66195 %EX The locale's alternative time representation.
- 66196 %Ey The offset from %EC (year only) in the locale's alternative representation.
- 66197 %EY The full alternative year representation.
- 66198 %Od The day of the month using the locale's alternative numeric symbols; leading zeros
66199 shall be permitted but shall not be required.
- 66200 %Oe Equivalent to %Od.
- 66201 %OH The hour (24-hour clock) using the locale's alternative numeric symbols.
- 66202 %OI The hour (12-hour clock) using the locale's alternative numeric symbols.
- 66203 %Om The month using the locale's alternative numeric symbols.
- 66204 %OM The minutes using the locale's alternative numeric symbols.
- 66205 %OS The seconds using the locale's alternative numeric symbols.
- 66206 %OU The week number of the year (Sunday as the first day of the week) using the locale's
66207 alternative numeric symbols.
- 66208 %Ow The number of the weekday (Sunday=0) using the locale's alternative numeric
66209 symbols.
- 66210 %OW The week number of the year (Monday as the first day of the week) using the locale's
66211 alternative numeric symbols.
- 66212 %Oy The year (offset from %C) using the locale's alternative numeric symbols.

66213 A conversion specification composed of white-space characters is executed by scanning input up
66214 to the first character that is not white-space (which remains unscanned), or until no more
66215 characters can be scanned.

66216 A conversion specification that is an ordinary character is executed by scanning the next
66217 character from the buffer. If the character scanned from the buffer differs from the one
66218 comprising the directive, the directive fails, and the differing and subsequent characters remain

66219 unscanned.

66220 A series of conversion specifications composed of %n, %t, white-space characters, or any
66221 combination is executed by scanning up to the first character that is not white space (which
66222 remains unscanned), or until no more characters can be scanned.

66223 Any other conversion specification is executed by scanning characters until a character matching
66224 the next directive is scanned, or until no more characters can be scanned. These characters,
66225 except the one matching the next directive, are then compared to the locale values associated
66226 with the conversion specifier. If a match is found, values for the appropriate **tm** structure
66227 members are set to values corresponding to the locale information. Case is ignored when
66228 matching items in *buf* such as month or weekday names. If no match is found, *strptime()* fails
66229 and no more characters are scanned.

66230 RETURN VALUE

66231 Upon successful completion, *strptime()* shall return a pointer to the character following the last
66232 character parsed. Otherwise, a null pointer shall be returned.

66233 ERRORS

66234 No errors are defined.

66235 EXAMPLES

66236 Convert a Date-Plus-Time String to Broken-Down Time and Then into Seconds

66237 The following example demonstrates the use of *strptime()* to convert a string into broken-down
66238 time. The broken-down time is then converted into seconds since the Epoch using *mktime()*.

```
66239 #include <time.h>
66240 ...
66241 struct tm tm;
66242 time_t t;
66243 if (strptime("6 Dec 2001 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
66244     /* Handle error */;
66245 printf("year: %d; month: %d; day: %d;\n",
66246        tm.tm_year, tm.tm_mon, tm.tm_mday);
66247 printf("hour: %d; minute: %d; second: %d\n",
66248        tm.tm_hour, tm.tm_min, tm.tm_sec);
66249 printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);
66250 tm.tm_isdst = -1;      /* Not set by strptime(); tells mktime()
66251                        to determine whether daylight saving time
66252                        is in effect */
66253 t = mktime(&tm);
66254 if (t == -1)
66255     /* Handle error */;
66256 printf("seconds since the Epoch: %ld\n", (long) t);"
```

66257 APPLICATION USAGE

66258 Several ``equivalent to'' formats and the special processing of white-space characters are
66259 provided in order to ease the use of identical *format* strings for *strptime()* and *strptime()*.

66260 It should be noted that dates constructed by the *strptime()* function with the %Y or %C%y
66261 conversion specifiers may have values larger than 9999. If the *strptime()* function is used to read
66262 such values using %C%y or %Y, the year values will be truncated to four digits. Applications

66263 should use `%+w%y` or `%+xY` with *w* and *x* set large enough to contain the full value of any years
66264 that will be printed or scanned.

66265 See also the APPLICATION USAGE section in *strptime()*.

66266 It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the
66267 current contents of the structure or overwrite all contents of the structure. Conforming
66268 applications should make a single call to *strptime()* with a format and all data needed to
66269 completely specify the date and time being converted.

66270 RATIONALE

66271 See the RATIONALE section for *strptime()*.

66272 FUTURE DIRECTIONS

66273 None.

66274 SEE ALSO

66275 *fprintf()*, *fscanf()*, *strptime()*, *time()*

66276 XBD [<time.h>](#)

66277 CHANGE HISTORY

66278 First released in Issue 4.

66279 Issue 5

66280 Moved from ENHANCED I18N to BASE.

66281 The [ENOSYS] error is removed.

66282 The exact meaning of the `%y` and `%0y` specifiers is clarified in the DESCRIPTION.

66283 Issue 6

66284 The Open Group Corrigendum U033/5 is applied. The `%r` specifier description is reworded.

66285 The normative text is updated to avoid use of the term “must” for application requirements.

66286 The **restrict** keyword is added to the *strptime()* prototype for alignment with the
66287 ISO/IEC 9899:1999 standard.

66288 The Open Group Corrigendum U047/2 is applied.

66289 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
66290 specification” for consistency with *strptime()*.

66291 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/133 is applied, adding the example to the
66292 EXAMPLES section.

66293 Issue 7

66294 Austin Group Interpretation 1003.1-2001 #041 is applied, updating the DESCRIPTION and
66295 APPLICATION USAGE sections.

66296 Austin Group Interpretation 1003.1-2001 #163 is applied.

66297 SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression
66298 that `%Y` is 4-digit years.

66299 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0345 [920] and XSH/TC2-2008/0346
66300 [919] are applied.

66301 **NAME**

66302 strrchr — string scanning operation

66303 **SYNOPSIS**

66304 #include <string.h>

66305 char *strrchr(const char *s, int c);

66306 **DESCRIPTION**

66307 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
66308 conflict between the requirements described here and the ISO C standard is unintentional. This
66309 volume of POSIX.1-2017 defers to the ISO C standard.

66310 The *strrchr()* function shall locate the last occurrence of *c* (converted to a **char**) in the string
66311 pointed to by *s*. The terminating NUL character is considered to be part of the string.

66312 **RETURN VALUE**

66313 Upon successful completion, *strrchr()* shall return a pointer to the byte or a null pointer if *c* does
66314 not occur in the string.

66315 **ERRORS**

66316 No errors are defined.

66317 **EXAMPLES**66318 **Finding the Base Name of a File**

66319 The following example uses *strrchr()* to get a pointer to the base name of a file. The *strrchr()*
66320 function searches backwards through the name of the file to find the last '/' character in *name*.
66321 This pointer (plus one) will point to the base name of the file.

```
66322       #include <string.h>
66323       ...
66324       const char *name;
66325       char *basename;
66326       ...
66327       basename = strrchr(name, '/') + 1;
66328       ...
```

66329 **APPLICATION USAGE**

66330 None.

66331 **RATIONALE**

66332 None.

66333 **FUTURE DIRECTIONS**

66334 None.

66335 **SEE ALSO**66336 [strchr\(\)](#)66337 XBD [<string.h>](#)66338 **CHANGE HISTORY**

66339 First released in Issue 1. Derived from Issue 1 of the SVID.

66340 **NAME**

66341 strsignal ‡get name of signal

66342 **SYNOPSIS**

```
66343 CX #include <string.h>
66344 char *strsignal(int signum);
```

66345 **DESCRIPTION**

66346 The *strsignal()* function shall map the signal number in *signum* to an implementation-defined
 66347 string and shall return a pointer to it. It shall use the same set of messages as the *psignal()*
 66348 function.

66349 The application shall not modify the string returned. The returned pointer might be invalidated
 66350 or the string content might be overwritten by a subsequent call to *strsignal()* or *setlocale()*. The
 66351 returned pointer might also be invalidated if the calling thread is terminated.

66352 The contents of the message strings returned by *strsignal()* should be determined by the setting
 66353 of the *LC_MESSAGES* category in the current locale.

66354 The implementation shall behave as if no function defined in this standard calls *strsignal()*.

66355 Since no return value is reserved to indicate an error, an application wishing to check for error
 66356 situations should set *errno* to 0, then call *strsignal()*, then check *errno*.

66357 The *strsignal()* function need not be thread-safe.

66358 **RETURN VALUE**

66359 Upon successful completion, *strsignal()* shall return a pointer to a string. Otherwise, if *signum* is
 66360 not a valid signal number, the return value is unspecified.

66361 **ERRORS**

66362 No errors are defined.

66363 **EXAMPLES**

66364 None.

66365 **APPLICATION USAGE**

66366 None.

66367 **RATIONALE**

66368 If *signum* is not a valid signal number, some implementations return NULL, while for others the
 66369 *strsignal()* function returns a pointer to a string containing an unspecified message denoting an
 66370 unknown signal. POSIX.1-2017 leaves this return value unspecified.

66371 **FUTURE DIRECTIONS**

66372 None.

66373 **SEE ALSO**

66374 *psiginfo()*, *setlocale()*

66375 XBD <string.h>

66376 **CHANGE HISTORY**

66377 First released in Issue 7.

66378 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0609 [75] is applied.

66379 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0347 [656] is applied.

66380 **NAME**

66381 strspn †'get length of a substring

66382 **SYNOPSIS**

66383 #include <string.h>

66384 size_t strspn(const char *s1, const char *s2);

66385 **DESCRIPTION**

66386 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
66387 conflict between the requirements described here and the ISO C standard is unintentional. This
66388 volume of POSIX.1-2017 defers to the ISO C standard.

66389 The *strspn()* function shall compute the length (in bytes) of the maximum initial segment of the
66390 string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

66391 **RETURN VALUE**

66392 The *strspn()* function shall return the computed length; no return value is reserved to indicate an
66393 error.

66394 **ERRORS**

66395 No errors are defined.

66396 **EXAMPLES**

66397 None.

66398 **APPLICATION USAGE**

66399 None.

66400 **RATIONALE**

66401 None.

66402 **FUTURE DIRECTIONS**

66403 None.

66404 **SEE ALSO**66405 [strcspn\(\)](#)66406 XBD [<string.h>](#)66407 **CHANGE HISTORY**

66408 First released in Issue 1. Derived from Issue 1 of the SVID.

66409 **Issue 5**

66410 The RETURN VALUE section is updated to indicate that *strspn()* returns the length of *s*, and not
66411 *s* itself as was previously stated.

66412 **Issue 7**

66413 SD5-XSH-ERN-182 is applied.

66414 **NAME**

66415 strstr †'find a substring

66416 **SYNOPSIS**

66417 #include <string.h>

66418 char *strstr(const char *s1, const char *s2);

66419 **DESCRIPTION**

66420 CX The functionality described on this reference page is aligned with the ISO C standard. Any
66421 conflict between the requirements described here and the ISO C standard is unintentional. This
66422 volume of POSIX.1-2017 defers to the ISO C standard.

66423 The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the
66424 sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.

66425 **RETURN VALUE**

66426 Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer
66427 if the string is not found.

66428 If *s2* points to a string with zero length, the function shall return *s1*.

66429 **ERRORS**

66430 No errors are defined.

66431 **EXAMPLES**

66432 None.

66433 **APPLICATION USAGE**

66434 None.

66435 **RATIONALE**

66436 None.

66437 **FUTURE DIRECTIONS**

66438 None.

66439 **SEE ALSO**66440 [strchr\(\)](#)66441 XBD [<string.h>](#)66442 **CHANGE HISTORY**

66443 First released in Issue 3. Included for alignment with the ANSI C standard.

66444 **NAME**

66445 strtod, strtodf, strtold — convert a string to a double-precision number

66446 **SYNOPSIS**

66447 #include <stdlib.h>

66448 double strtod(const char *restrict *nptr*, char **restrict *endptr*);66449 float strtodf(const char *restrict *nptr*, char **restrict *endptr*);66450 long double strtold(const char *restrict *nptr*, char **restrict *endptr*);66451 **DESCRIPTION**

66452 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 66453 conflict between the requirements described here and the ISO C standard is unintentional. This
 66454 volume of POSIX.1-2017 defers to the ISO C standard.

66455 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,
 66456 and **long double** representation, respectively. First, they decompose the input string into three
 66457 parts:

- 66458 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 66459 2. A subject sequence interpreted as a floating-point constant or representing infinity or
 66460 NaN
- 66461 3. A final string of one or more unrecognized characters, including the terminating NUL
 66462 character of the input string

66463 Then they shall attempt to convert the subject sequence to a floating-point number, and return
 66464 the result.

66465 The expected form of the subject sequence is an optional '+' or '-' sign, then one of the
 66466 following:

66467 A non-empty sequence of decimal digits optionally containing a radix character; then an
 66468 optional exponent part consisting of the character 'e' or the character 'E', optionally
 66469 followed by a '+' or '-' character, and then followed by one or more decimal digits

66470 A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix
 66471 character; then an optional binary exponent part consisting of the character 'p' or the
 66472 character 'P', optionally followed by a '+' or '-' character, and then followed by one or
 66473 more decimal digits

66474 One of INF or INFINITY, ignoring case

66475 One of NAN or NAN(*n-char-sequence_{opt}*), ignoring case in the NAN part, where:

66476 n-char-sequence:

66477 digit

66478 nondigit

66479 n-char-sequence digit

66480 n-char-sequence nondigit

66481 The subject sequence is defined as the longest initial subsequence of the input string, starting
 66482 with the first non-white-space character, that is of the expected form. The subject sequence
 66483 contains no characters if the input string is not of the expected form.

66484 If the subject sequence has the expected form for a floating-point number, the sequence of
 66485 characters starting with the first digit or the decimal-point character (whichever occurs first)
 66486 shall be interpreted as a floating constant of the C language, except that the radix character shall
 66487 be used in place of a period, and that if neither an exponent part nor a radix character appears in

66488 a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal
 66489 floating-point number, an exponent part of the appropriate type with value zero is assumed to
 66490 follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the
 66491 sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be
 66492 interpreted as an infinity, if representable in the return type, else as if it were a floating constant
 66493 that is too large for the range of the return type. A character sequence NAN or NAN(*n-char-*
 66494 *sequence_{opt}*) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a
 66495 subject sequence part that does not have the expected form; the meaning of the *n-char* sequences
 66496 is implementation-defined. A pointer to the final string is stored in the object pointed to by
 66497 *endptr*, provided that *endptr* is not a null pointer.

66498 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value
 66499 resulting from the conversion is correctly rounded.

66500 CX The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX
 66501 locale, or in a locale where the radix character is not defined, the radix character shall default to
 66502 a <period> ('.').

66503 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
 66504 accepted.

66505 If the subject sequence is empty or does not have the expected form, no conversion shall be
 66506 performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is
 66507 not a null pointer.

66508 These functions shall not change the setting of *errno* if successful.

66509 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 66510 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

66511 RETURN VALUE

66512 Upon successful completion, these functions shall return the converted value. If no conversion
 66513 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

66514 If the correct value is outside the range of representable values, \pm HUGE_VAL, \pm HUGE_VALF, or
 66515 \pm HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to
 66516 [ERANGE].

66517 If the correct value would cause an underflow, a value whose magnitude is no greater than the
 66518 smallest normalized positive number in the return type shall be returned and *errno* set to
 66519 [ERANGE].

66520 ERRORS

66521 These functions shall fail if:

66522 CX [ERANGE] The value to be returned would cause overflow or underflow.

66523 These functions may fail if:

66524 CX [EINVAL] No conversion could be performed.

66525 **EXAMPLES**

66526 None.

66527 **APPLICATION USAGE**

66528 If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the
 66529 result is not exactly representable, the result should be one of the two numbers in the
 66530 appropriate internal format that are adjacent to the hexadecimal floating source value, with the
 66531 extra stipulation that the error should have a correct sign for the current rounding direction.

66532 If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in `<float.h>`)
 66533 significant digits, the result should be correctly rounded. If the subject sequence *D* has the
 66534 decimal form and more than DECIMAL_DIG significant digits, consider the two bounding,
 66535 adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the
 66536 values of *L*, *D*, and *U* satisfy $L \leq D \leq U$. The result should be one of the (equal or adjacent)
 66537 values that would be obtained by correctly rounding *L* and *U* according to the current rounding
 66538 direction, with the extra stipulation that the error with respect to *D* should have a correct sign
 66539 for the current rounding direction.

66540 The changes to `strtod()` introduced by the ISO/IEC 9899:1999 standard can alter the behavior of
 66541 well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier
 66542 versions of this standard. One such example would be:

```

66543 int
66544 what_kind_of_number (char *s)
66545 {
66546     char *endp;
66547     double d;
66548     long l;
66549
66549     d = strtod(s, &endp);
66550     if (s != endp && *endp == '\0')
66551         printf("It's a float with value %g\n", d);
66552     else
66553     {
66554         l = strtol(s, &endp, 0);
66555         if (s != endp && *endp == '\0')
66556             printf("It's an integer with value %ld\n", l);
66557         else
66558             return 1;
66559     }
66560     return 0;
66561 }

```

66562 If the function is called with:

```
66563 what_kind_of_number ("0x10")
```

66564 an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
66565 It's an integer with value 16
```

66566 With the ISO/IEC 9899:1999 standard, the result is:

```
66567 It's a float with value 16
```

66568 The change in behavior is due to the inclusion of floating-point numbers in hexadecimal
 66569 notation without requiring that either a decimal point or the binary exponent be present.

66570 RATIONALE

66571 None.

66572 FUTURE DIRECTIONS

66573 None.

66574 SEE ALSO

66575 *fscanf()*, *isspace()*, *localeconv()*, *setlocale()*, *strtol()*

66576 XBD Chapter 7 (on page 135), [<float.h>](#), [<stdlib.h>](#)

66577 CHANGE HISTORY

66578 First released in Issue 1. Derived from Issue 1 of the SVID.

66579 Issue 5

66580 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

66581 Issue 6

66582 Extensions beyond the ISO C standard are marked.

66583 The following new requirements on POSIX implementations derive from alignment with the
66584 Single UNIX Specification:

66585 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
66586 added if no conversion could be performed.

66587 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

66588 The *strtod()* function is updated.

66589 The *strtof()* and *strtold()* functions are added.

66590 The DESCRIPTION is extensively revised.

66591 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

66592 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second
66593 paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.

66594 Issue 7

66595 Austin Group Interpretation 1003.1-2001 #015 is applied.

66596 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0610 [302], XSH/TC1-2008/0611 [94],
66597 and XSH/TC1-2008/0612 [105] are applied.

66598 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0348 [584] and XSH/TC2-2008/0349
66599 [796] are applied.

66600 **NAME**

66601 strtoimax, strtoumax ‡convert string to integer type

66602 **SYNOPSIS**

66603 #include <inttypes.h>

66604 intmax_t strtoimax(const char *restrict *nptr*, char **restrict *endptr*,
66605 int *base*);66606 uintmax_t strtoumax(const char *restrict *nptr*, char **restrict *endptr*,
66607 int *base*);66608 **DESCRIPTION**66609 CX The functionality described on this reference page is aligned with the ISO C standard. Any
66610 conflict between the requirements described here and the ISO C standard is unintentional. This
66611 volume of POSIX.1-2017 defers to the ISO C standard.66612 These functions shall be equivalent to the *strtol()*, *strtoll()*, *strtoul()*, and *strtoull()* functions,
66613 except that the initial portion of the string shall be converted to **intmax_t** and **uintmax_t**
66614 representation, respectively.66615 **RETURN VALUE**

66616 These functions shall return the converted value, if any.

66617 CX If no conversion could be performed, zero shall be returned and *errno* may be set to [EINVAL].66618 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].66619 If the correct value is outside the range of representable values, {INTMAX_MAX},
66620 {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign
66621 of the value, if any), and *errno* shall be set to [ERANGE].66622 **ERRORS**

66623 These functions shall fail if:

66624 CX [EINVAL] The value of *base* is not supported.

66625 [ERANGE] The value to be returned is not representable.

66626 These functions may fail if:

66627 [EINVAL] No conversion could be performed.

66628 **EXAMPLES**

66629 None.

66630 **APPLICATION USAGE**66631 Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should
66632 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for
66633 an [EINVAL] error before examining **endptr*.66634 **RATIONALE**

66635 None.

66636 **FUTURE DIRECTIONS**

66637 None.

66638 **SEE ALSO**66639 [strtol\(\)](#), [strtoul\(\)](#)66640 XBD [<inttypes.h>](#)

66641 **CHANGE HISTORY**

66642 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

66643 **Issue 7**

66644 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0613 [453] and XSH/TC1-2008/0614
66645 [453] are applied.

66646 **NAME**

66647 strtok, strtok_r †split string into tokens

66648 **SYNOPSIS**

66649 #include <string.h>

66650 char *strtok(char *restrict *s*, const char *restrict *sep*);66651 CX char *strtok_r(char *restrict *s*, const char *restrict *sep*,66652 char **restrict *state*);66653 **DESCRIPTION**66654 CX For *strtok()*: The functionality described on this reference page is aligned with the ISO C
66655 standard. Any conflict between the requirements described here and the ISO C standard is
66656 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.66657 A sequence of calls to *strtok()* breaks the string pointed to by *s* into a sequence of tokens, each of
66658 which is delimited by a byte from the string pointed to by *sep*. The first call in the sequence has *s*
66659 as its first argument, and is followed by calls with a null pointer as their first argument. The
66660 separator string pointed to by *sep* may be different from call to call.66661 The first call in the sequence searches the string pointed to by *s* for the first byte that is *not*
66662 contained in the current separator string pointed to by *sep*. If no such byte is found, then there
66663 are no tokens in the string pointed to by *s* and *strtok()* shall return a null pointer. If such a byte is
66664 found, it is the start of the first token.66665 The *strtok()* function then searches from there for a byte that *is* contained in the current separator
66666 string. If no such byte is found, the current token extends to the end of the string pointed to by *s*,
66667 and subsequent searches for a token shall return a null pointer. If such a byte is found, it is
66668 overwritten by a NUL character, which terminates the current token. The *strtok()* function saves
66669 a pointer to the following byte, from which the next search for a token shall start.66670 Each subsequent call, with a null pointer as the value of the first argument, starts searching from
66671 the saved pointer and behaves as described above.66672 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
66673 *strtok()*.66674 CX The *strtok()* function need not be thread-safe.66675 The *strtok_r()* function shall be equivalent to *strtok()*, except that *strtok_r()* shall be thread-safe
66676 and the argument *state* points to a user-provided pointer that allows *strtok_r()* to maintain state
66677 between calls which scan the same string. The application shall ensure that the pointer pointed
66678 to by *state* is unique for each string (*s*) being processed concurrently by *strtok_r()* calls. The
66679 application need not initialize the pointer pointed to by *state* to any particular value. The
66680 implementation shall not update the pointer pointed to by *state* to point (directly or indirectly) to
66681 resources, other than within the string *s*, that need to be freed or released by the caller.66682 **RETURN VALUE**66683 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,
66684 if there is no token, *strtok()* shall return a null pointer.66685 CX The *strtok_r()* function shall return a pointer to the token found, or a null pointer when no token
66686 is found.

66687 **ERRORS**

66688 No errors are defined.

66689 **EXAMPLES**66690 **Searching for Word Separators**

66691 The following example searches for tokens separated by <space> characters.

```

66692 #include <string.h>
66693 ...
66694 char *token;
66695 char line[] = "LINE TO BE SEPARATED";
66696 char *search = " ";

66697 /* Token will point to "LINE". */
66698 token = strtok(line, search);

66699 /* Token will point to "TO". */
66700 token = strtok(NULL, search);

```

66701 **Find First two Fields in a Buffer**

66702 The following example uses *strtok()* to find two character strings (a key and data associated with
 66703 that key) separated by any combination of <space>, <tab>, or <newline> characters at the start
 66704 of the array of characters pointed to by *buffer*.

```

66705 #include <string.h>
66706 ...
66707 char    *buffer;
66708 ...
66709 struct element {
66710     char *key;
66711     char *data;
66712 } e;
66713 ...
66714 // Load the buffer...
66715 ...
66716 // Get the key and its data...
66717 e.key = strtok(buffer, " \t\n");
66718 e.data = strtok(NULL, " \t\n");
66719 // Process the rest of the contents of the buffer...
66720 ...

```

66721 **APPLICATION USAGE**

66722 Note that if *sep* is the empty string, *strtok()* and *strtok_r()* return a pointer to the remainder of the
 66723 string being tokenized.

66724 The *strtok_r()* function is thread-safe and stores its state in a user-supplied buffer instead of
 66725 possibly using a static data area that may be overwritten by an unrelated call from another
 66726 thread.

66727 **RATIONALE**

66728 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to
 66729 the last substring between separator strings. This function uses static storage to keep track of
 66730 the current string position between calls. The new function, *strtok_r()*, takes an additional

- 66731 argument, *state*, to keep track of the current position in the string.
- 66732 **FUTURE DIRECTIONS**
- 66733 None.
- 66734 **SEE ALSO**
- 66735 XBD <[string.h](#)>
- 66736 **CHANGE HISTORY**
- 66737 First released in Issue 1. Derived from Issue 1 of the SVID.
- 66738 **Issue 5**
- 66739 The *strtok_r()* function is included for alignment with the POSIX Threads Extension.
- 66740 A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.
- 66741 **Issue 6**
- 66742 Extensions beyond the ISO C standard are marked.
- 66743 The *strtok_r()* function is marked as part of the Thread-Safe Functions option.
- 66744 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.
- 66745 The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.
- 66746
- 66747 The **restrict** keyword is added to the *strtok()* and *strtok_r()* prototypes for alignment with the
- 66748 ISO/IEC 9899:1999 standard.
- 66749 **Issue 7**
- 66750 Austin Group Interpretation 1003.1-2001 #156 is applied.
- 66751 SD5-XSH-ERN-235 is applied, correcting an example.
- 66752 The *strtok_r()* function is moved from the Thread-Safe Functions option to the Base.
- 66753 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0615 [177] is applied.
- 66754 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0350 [878] is applied.

66755 **NAME**66756 `strtol, strtoll` ¶convert a string to a long integer66757 **SYNOPSIS**66758 `#include <stdlib.h>`66759 `long strtol(const char *restrict nptr, char **restrict endptr, int base);`66760 `long long strtoll(const char *restrict nptr, char **restrict endptr,`66761 `int base)`66762 **DESCRIPTION**

66763 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 66764 conflict between the requirements described here and the ISO C standard is unintentional. This
 66765 volume of POSIX.1-2017 defers to the ISO C standard.

66766 These functions shall convert the initial portion of the string pointed to by *nptr* to a type **long**
 66767 and **long long** representation, respectively. First, they decompose the input string into three
 66768 parts:

- 66769 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 66770 2. A subject sequence interpreted as an integer represented in some radix determined by the
 66771 value of *base*
- 66772 3. A final string of one or more unrecognized characters, including the terminating NUL
 66773 character of the input string.

66774 Then they shall attempt to convert the subject sequence to an integer, and return the result.

66775 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
 66776 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
 66777 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
 66778 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
 66779 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 66780 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

66781 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 66782 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 66783 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
 66784 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
 66785 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
 66786 digits, following the sign if present.

66787 The subject sequence is defined as the longest initial subsequence of the input string, starting
 66788 with the first non-white-space character that is of the expected form. The subject sequence shall
 66789 contain no characters if the input string is empty or consists entirely of white-space characters,
 66790 or if the first non-white-space character is other than a sign or a permissible letter or digit.

66791 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
 66792 characters starting with the first digit shall be interpreted as an integer constant. If the subject
 66793 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
 66794 base for conversion, ascribing to each letter its value as given above. If the subject sequence
 66795 begins with a <hyphen-minus>, the value resulting from the conversion shall be negated. A
 66796 pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr*
 66797 is not a null pointer.

66798 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
 66799 accepted.

66800 If the subject sequence is empty or does not have the expected form, no conversion is performed;
 66801 the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a
 66802 null pointer.

66803 These functions shall not change the setting of *errno* if successful.

66804 Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned
 66805 on error and are also valid returns on success, an application wishing to check for error
 66806 situations should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

66807 RETURN VALUE

66808 Upon successful completion, these functions shall return the converted value, if any. If no
 66809 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

66810 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

66811 If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
 66812 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
 66813 *errno* set to [ERANGE].

66814 ERRORS

66815 These functions shall fail if:

66816 CX [EINVAL] The value of *base* is not supported.

66817 [ERANGE] The value to be returned is not representable.

66818 These functions may fail if:

66819 [EINVAL] No conversion could be performed.

66820 EXAMPLES

66821 None.

66822 APPLICATION USAGE

66823 Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should
 66824 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for
 66825 an [EINVAL] error before examining **endptr*.

66826 RATIONALE

66827 None.

66828 FUTURE DIRECTIONS

66829 None.

66830 SEE ALSO

66831 *fscanf()*, *isalpha()*, *strtod()*

66832 XBD <stdlib.h>

66833 CHANGE HISTORY

66834 First released in Issue 1. Derived from Issue 1 of the SVID.

66835 Issue 5

66836 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

66837 Issue 6

66838 Extensions beyond the ISO C standard are marked.

66839 The following new requirements on POSIX implementations derive from alignment with the
 66840 Single UNIX Specification:

66841 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
66842 added if no conversion could be performed.

66843 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

66844 The *strtol()* prototype is updated.

66845 The *strtoll()* function is added.

66846 **Issue 7**

66847 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0616 [453], XSH/TC1-2008/0617 [105],
66848 XSH/TC1-2008/0618 [453], XSH/TC1-2008/0619 [453], and XSH/TC1-2008/0620 [453] are
66849 applied.

66850 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0351 [892], XSH/TC2-2008/0352 [584],
66851 XSH/TC2-2008/0353 [796], and XSH/TC2-2008/0354 [892] are applied.

66852 **NAME**

66853 strtold — convert a string to a double-precision number

66854 **SYNOPSIS**

66855 #include <stdlib.h>

66856 long double strtold(const char *restrict *nptr*, char **restrict *endptr*);

66857 **DESCRIPTION**

66858 Refer to *strtod()*.

66859 **NAME**

66860 strtoll ¶convert a string to a long integer

66861 **SYNOPSIS**

66862 #include <stdlib.h>

66863 long long strtoll(const char *restrict *str*, char **restrict *endptr*,
66864 int *base*);66865 **DESCRIPTION**66866 Refer to *strtol()*.

66867 **NAME**

66868 strtol, strtoull ‡convert a string to an unsigned long

66869 **SYNOPSIS**

66870 #include <stdlib.h>

```
66871            unsigned long strtol(const char *restrict str,
66872                                  char **restrict endptr, int base);
66873            unsigned long long strtoull(const char *restrict str,
66874                                        char **restrict endptr, int base);
```

66875 **DESCRIPTION**

66876 CX The functionality described on this reference page is aligned with the ISO C standard. Any
66877 conflict between the requirements described here and the ISO C standard is unintentional. This
66878 volume of POSIX.1-2017 defers to the ISO C standard.

66879 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned
66880 long** and **unsigned long long** representation, respectively. First, they decompose the input string
66881 into three parts:

- 66882 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 66883 2. A subject sequence interpreted as an integer represented in some radix determined by the
66884 value of *base*
- 66885 3. A final string of one or more unrecognized characters, including the terminating NUL
66886 character of the input string

66887 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
66888 result.

66889 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
66890 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
66891 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
66892 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
66893 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
66894 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

66895 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
66896 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
66897 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
66898 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
66899 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
66900 digits, following the sign if present.

66901 The subject sequence is defined as the longest initial subsequence of the input string, starting
66902 with the first non-white-space character that is of the expected form. The subject sequence shall
66903 contain no characters if the input string is empty or consists entirely of white-space characters,
66904 or if the first non-white-space character is other than a sign or a permissible letter or digit.

66905 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
66906 characters starting with the first digit shall be interpreted as an integer constant. If the subject
66907 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
66908 base for conversion, ascribing to each letter its value as given above. If the subject sequence
66909 begins with a <hyphen-minus>, the value resulting from the conversion shall be negated. A
66910 pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr*
66911 is not a null pointer.

66912 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
66913 accepted.

66914 If the subject sequence is empty or does not have the expected form, no conversion shall be
66915 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*
66916 is not a null pointer.

66917 These functions shall not change the setting of *errno* if successful.

66918 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns
66919 on success, an application wishing to check for error situations should set *errno* to 0, then call
66920 *strtoul()* or *strtoull()*, then check *errno*.

66921 RETURN VALUE

66922 Upon successful completion, these functions shall return the converted value, if any. If no
66923 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

66924 CX If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to [EINVAL].

66925 If the correct value is outside the range of representable values, {ULONG_MAX} or
66926 {ULLONG_MAX} shall be returned and *errno* set to [ERANGE].

66927 ERRORS

66928 These functions shall fail if:

66929 CX [EINVAL] The value of *base* is not supported.

66930 [ERANGE] The value to be returned is not representable.

66931 These functions may fail if:

66932 CX [EINVAL] No conversion could be performed.

66933 EXAMPLES

66934 None.

66935 APPLICATION USAGE

66936 Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should
66937 either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for
66938 an [EINVAL] error before examining **endptr*.

66939 RATIONALE

66940 None.

66941 FUTURE DIRECTIONS

66942 None.

66943 SEE ALSO

66944 *fscanf()*, *isalpha()*, *strtod()*, *strtol()*

66945 XBD <stdlib.h>

66946 CHANGE HISTORY

66947 First released in Issue 4. Derived from the ANSI C standard.

66948 Issue 5

66949 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

66950 **Issue 6**

66951 Extensions beyond the ISO C standard are marked.

66952 The following new requirements on POSIX implementations derive from alignment with the
66953 Single UNIX Specification:

66954 The [EINVAL] error condition is added for when the value of *base* is not supported.

66955 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
66956 added if no conversion could be performed.

66957 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

66958 The *strtol()* prototype is updated.

66959 The *strtoll()* function is added.

66960 **Issue 7**

66961 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0621 [105], XSH/TC1-2008/0622 [453],
66962 and XSH/TC1-2008/0623 [453] are applied.

66963 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0355 [584] and XSH/TC2-2008/0356
66964 [796] are applied.

66965 **NAME**

66966 strtoumax ‡convert a string to an integer type

66967 **SYNOPSIS**

66968 #include <inttypes.h>

66969 uintmax_t strtoumax(const char *restrict *nptr*, char **restrict *endptr*,
66970 int *base*);66971 **DESCRIPTION**66972 Refer to *strtoimax()*.

66973 **NAME**

66974 strxfrm, strxfrm_l †string transformation

66975 **SYNOPSIS**

66976 #include <string.h>

66977 size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);

66978 CX size_t strxfrm_l(char *restrict s1, const char *restrict s2,

66979 size_t n, locale_t locale);

66980 **DESCRIPTION**66981 CX For *strxfrm()*: The functionality described on this reference page is aligned with the ISO C
66982 standard. Any conflict between the requirements described here and the ISO C standard is
66983 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.66984 CX The *strxfrm()* and *strxfrm_l()* functions shall transform the string pointed to by *s2* and place the
66985 resulting string into the array pointed to by *s1*. The transformation is such that if *strcmp()* is
66986 applied to two transformed strings, it shall return a value greater than, equal to, or less than 0,
66987 CX corresponding to the result of *strcoll()* or *strcoll_l()*, respectively, applied to the same two
66988 CX original strings with the same locale. No more than *n* bytes are placed into the resulting array
66989 pointed to by *s1*, including the terminating NUL character. If *n* is 0, *s1* is permitted to be a null
66990 pointer. If copying takes place between objects that overlap, the behavior is undefined.66991 CX The *strxfrm()* and *strxfrm_l()* functions shall not change the setting of *errno* if successful.66992 Since no return value is reserved to indicate an error, an application wishing to check for error
66993 CX situations should set *errno* to 0, then call *strxfrm()* or *strxfrm_l()*, then check *errno*.66994 CX The behavior is undefined if the *locale* argument to *strxfrm_l()* is the special locale object
66995 LC_GLOBAL_LOCALE or is not a valid locale object handle.66996 **RETURN VALUE**66997 CX Upon successful completion, *strxfrm()* and *strxfrm_l()* shall return the length of the
66998 transformed string (not including the terminating NUL character). If the value returned is *n* or
66999 more, the contents of the array pointed to by *s1* are unspecified.67000 CX On error, *strxfrm()* and *strxfrm_l()* may set *errno* but no return value is reserved to indicate an
67001 error.67002 **ERRORS**

67003 These functions may fail if:

67004 CX [EINVAL] The string pointed to by the *s2* argument contains characters outside the
67005 domain of the collating sequence.67006 **EXAMPLES**

67007 None.

67008 **APPLICATION USAGE**67009 The transformation function is such that two transformed strings can be ordered by *strcmp()* as
67010 appropriate to collating sequence information in the current locale (category *LC_COLLATE*).67011 The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the
67012 *s1* array prior to making the transformation.

67013 **RATIONALE**

67014 None.

67015 **FUTURE DIRECTIONS**

67016 None.

67017 **SEE ALSO**67018 [strcmp\(\)](#), [strcoll\(\)](#)67019 XBD [<string.h>](#)67020 **CHANGE HISTORY**

67021 First released in Issue 3. Included for alignment with the ISO C standard.

67022 **Issue 5**67023 The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.67024 **Issue 6**

67025 Extensions beyond the ISO C standard are marked.

67026 The following new requirements on POSIX implementations derive from alignment with the
67027 Single UNIX Specification:67028 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
67029 added if no conversion could be performed.67030 The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.67031 **Issue 7**67032 The *strxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended API
67033 Set Part 4.67034 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0624 [283], XSH/TC1-2008/0625 [283],
67035 and XSH/TC1-2008/0626 [302] are applied.

67036 **NAME**

67037 swab ‡swap bytes

67038 **SYNOPSIS**

```
67039 XSI #include <unistd.h>
67040 void swab(const void *restrict src, void *restrict dest,
67041           ssize_t nbytes);
```

67042 **DESCRIPTION**

67043 The *swab()* function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to
67044 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab()*
67045 copies and exchanges *nbytes*-1 bytes and the disposition of the last byte is unspecified. If
67046 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is
67047 negative, *swab()* does nothing.

67048 **RETURN VALUE**

67049 None.

67050 **ERRORS**

67051 No errors are defined.

67052 **EXAMPLES**

67053 None.

67054 **APPLICATION USAGE**

67055 None.

67056 **RATIONALE**

67057 None.

67058 **FUTURE DIRECTIONS**

67059 None.

67060 **SEE ALSO**67061 XBD <[unistd.h](#)>67062 **CHANGE HISTORY**

67063 First released in Issue 1. Derived from Issue 1 of the SVID.

67064 **Issue 6**

67065 The **restrict** keyword is added to the *swab()* prototype for alignment with the
67066 ISO/IEC 9899:1999 standard.

67067 **NAME**

67068 swprintf ‡'print formatted wide-character output

67069 **SYNOPSIS**

67070 #include <stdio.h>

67071 #include <wchar.h>

67072 int swprintf(wchar_t *restrict *ws*, size_t *n*,67073 const wchar_t *restrict *format*, ...);67074 **DESCRIPTION**67075 Refer to [fwprintf\(\)](#).

67076 **NAME**
67077 `swscanf` ‡'convert formatted wide-character input

67078 **SYNOPSIS**
67079 `#include <stdio.h>`
67080 `#include <wchar.h>`
67081 `int swscanf(const wchar_t *restrict ws,`
67082 `const wchar_t *restrict format, ...);`

67083 **DESCRIPTION**
67084 Refer to *fwscanf()*.

67085 **NAME**

67086 symlink, symlinkat ‡make a symbolic link

67087 **SYNOPSIS**

67088 #include <unistd.h>

67089 int symlink(const char *path1, const char *path2);

67090 OH #include <fcntl.h>

67091 int symlinkat(const char *path1, int fd, const char *path2);

67092 **DESCRIPTION**

67093 The *symlink()* function shall create a symbolic link called *path2* that contains the string pointed to
 67094 by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the
 67095 symbolic link).

67096 The string pointed to by *path1* shall be treated only as a string and shall not be validated as a
 67097 pathname.

67098 If the *symlink()* function fails for any reason other than [EIO], any file named by *path2* shall be
 67099 unaffected.

67100 If *path2* names a symbolic link, *symlink()* shall fail and set *errno* to [EEXIST].

67101 The symbolic link's user ID shall be set to the process' effective user ID. The symbolic link's
 67102 group ID shall be set to the group ID of the parent directory or to the effective group ID of the
 67103 process. Implementations shall provide a way to initialize the symbolic link's group ID to the
 67104 group ID of the parent directory. Implementations may, but need not, provide an
 67105 implementation-defined way to initialize the symbolic link's group ID to the effective group ID
 67106 of the calling process.

67107 The values of the file mode bits for the created symbolic link are unspecified. All interfaces
 67108 specified by POSIX.1-2017 shall behave as if the contents of symbolic links can always be read,
 67109 except that the value of the file mode bits returned in the *st_mode* field of the **stat** structure is
 67110 unspecified.

67111 Upon successful completion, *symlink()* shall mark for update the last data access, last data
 67112 modification, and last file status change timestamps of the symbolic link. Also, the last data
 67113 modification and last file status change timestamps of the directory that contains the new entry
 67114 shall be marked for update.

67115 The *symlinkat()* function shall be equivalent to the *symlink()* function except in the case where
 67116 *path2* specifies a relative path. In this case the symbolic link is created relative to the directory
 67117 associated with the file descriptor *fd* instead of the current working directory. If the access mode
 67118 of the open file description associated with the file descriptor is not O_SEARCH, the function
 67119 shall check whether directory searches are permitted using the current permissions of the
 67120 directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not
 67121 perform the check.

67122 If *symlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 67123 directory shall be used and the behavior shall be identical to a call to *symlink()*.

67124 **RETURN VALUE**

67125 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 67126 return -1 and set *errno* to indicate the error.

67127 **ERRORS**

67128 These functions shall fail if:

67129 [EACCES] Write permission is denied in the directory where the symbolic link is being
 67130 created, or search permission is denied for a component of the path prefix of
 67131 *path2*.

67132 [EEXIST] The *path2* argument names an existing file.

67133 [EIO] An I/O error occurs while reading from or writing to the file system.

67134 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path2*
 67135 argument.

67136 [ENAMETOOLONG]

67137 The length of a component of the pathname specified by the *path2* argument is
 67138 longer than {NAME_MAX} or the length of the *path1* argument is longer than
 67139 {SYMLINK_MAX}.

67140 [ENOENT] A component of the path prefix of *path2* does not name an existing file or *path2*
 67141 is an empty string.

67142 [ENOENT] or [ENOTDIR]

67143 The *path2* argument contains at least one non-`<slash>` character and ends with
 67144 one or more trailing `<slash>` characters. If *path2* without the trailing `<slash>`
 67145 characters would name an existing file, an [ENOENT] error shall not occur.

67146 [ENOSPC] The directory in which the entry for the new symbolic link is being placed
 67147 cannot be extended because no space is left on the file system containing the
 67148 directory, or the new symbolic link cannot be created because no space is left
 67149 on the file system which shall contain the link, or the file system is out of file-
 67150 allocation resources.

67151 [ENOTDIR] A component of the path prefix of *path2* names an existing file that is neither a
 67152 directory nor a symbolic link to a directory.

67153 [EROFS] The new symbolic link would reside on a read-only file system.

67154 The *symlinkat()* function shall fail if:

67155 [EACCES] The access mode of the open file description associated with *fd* is not
 67156 O_SEARCH and the permissions of the directory underlying *fd* do not permit
 67157 directory searches.

67158 [EBADF] The *path2* argument does not specify an absolute path and the *fd* argument is
 67159 neither AT_FDCWD nor a valid file descriptor open for reading or searching.

67160 [ENOTDIR] The *path2* argument is not an absolute path and *fd* is a file descriptor
 67161 associated with a non-directory file.

67162 These functions may fail if:

67163 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 67164 resolution of the *path2* argument.

67165 [ENAMETOOLONG]

67166 The length of the *path2* argument exceeds {PATH_MAX} or pathname
 67167 resolution of a symbolic link in the *path2* argument produced an intermediate
 67168 result with a length that exceeds {PATH_MAX}.

67169 EXAMPLES

67170 None.

67171 APPLICATION USAGE

67172 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a
67173 hard link guarantees the existence of a file, even after the original name has been removed. A
67174 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not
67175 exist when the link is created. A symbolic link can cross file system boundaries.

67176 Normal permission checks are made on each component of the symbolic link pathname during
67177 its resolution.

67178 RATIONALE

67179 The purpose of the *symlinkat()* function is to create symbolic links in directories other than the
67180 current working directory without exposure to race conditions. Any part of the path of a file
67181 could be changed in parallel to a call to *symlink()*, resulting in unspecified behavior. By opening
67182 a file descriptor for the target directory and using the *symlinkat()* function it can be guaranteed
67183 that the created symbolic link is located relative to the desired directory.

67184 FUTURE DIRECTIONS

67185 None.

67186 SEE ALSO

67187 *fdopendir()*, *fstatat()*, *lchown()*, *link()*, *open()*, *readlink()*, *rename()*, *unlink()*

67188 XBD [<fcntl.h>](#), [<unistd.h>](#)

67189 CHANGE HISTORY

67190 First released in Issue 4, Version 2.

67191 Issue 5

67192 Moved from X/OPEN UNIX extension to BASE.

67193 Issue 6

67194 The following changes were made to align with the IEEE P1003.1a draft standard:

67195 The DESCRIPTION text is updated.

67196 The [ELOOP] optional error condition is added.

67197 Issue 7

67198 Austin Group Interpretation 1003.1-2001 #143 is applied.

67199 The *symlinkat()* function is added from The Open Group Technical Standard, 2006, Extended
67200 API Set Part 2.

67201 Additions have been made describing how *symlink()* sets the user and group IDs and file mode
67202 of the symbolic link, and its effect on timestamps.

67203 Changes are made to allow a directory to be opened for searching.

67204 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0627 [146,428], XSH/TC1-2008/0628
67205 [461], XSH/TC1-2008/0629 [146,428], XSH/TC1-2008/0630 [146,428,436], XSH/TC1-2008/0631
67206 [324], XSH/TC1-2008/0632 [278], XSH/TC1-2008/0633 [278], and XSH/TC1-2008/0634 [151] are
67207 applied.

67208 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0357 [873], XSH/TC2-2008/0358 [591],
67209 XSH/TC2-2008/0359 [641], XSH/TC2-2008/0360 [817], XSH/TC2-2008/0361 [822],
67210 XSH/TC2-2008/0362 [817], and XSH/TC2-2008/0363 [591] are applied.

67211 **NAME**

67212 sync ‡schedule file system updates

67213 **SYNOPSIS**

67214 XSI #include <unistd.h>

67215 void sync(void);

67216 **DESCRIPTION**67217 The *sync()* function shall cause all information in memory that updates file systems to be
67218 scheduled for writing out to all file systems.67219 The writing, although scheduled, is not necessarily complete upon return from *sync()*.67220 **RETURN VALUE**67221 The *sync()* function shall not return a value.67222 **ERRORS**

67223 No errors are defined.

67224 **EXAMPLES**

67225 None.

67226 **APPLICATION USAGE**

67227 None.

67228 **RATIONALE**

67229 None.

67230 **FUTURE DIRECTIONS**

67231 None.

67232 **SEE ALSO**67233 [fsync\(\)](#)67234 XBD [<unistd.h>](#)67235 **CHANGE HISTORY**

67236 First released in Issue 4, Version 2.

67237 **Issue 5**

67238 Moved from X/OPEN UNIX extension to BASE.

67239 **NAME**

67240 sysconf ‡get configurable system variables

67241 **SYNOPSIS**

67242 #include <unistd.h>

67243 long sysconf(int name);

67244 **DESCRIPTION**

67245 The *sysconf()* function provides a method for the application to determine the current value of a
 67246 configurable system limit or option (*variable*). The implementation shall support all of the
 67247 variables listed in the following table and may support others.

67248 The *name* argument represents the system variable to be queried. The following table lists the
 67249 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,
 67250 and the symbolic constants defined in <unistd.h> that are the corresponding values used for
 67251 *name*.

67252	Variable	Value of Name
67253	{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
67254	{AIO_MAX}	_SC_AIO_MAX
67255	{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
67256	{ARG_MAX}	_SC_ARG_MAX
67257	{ATEXIT_MAX}	_SC_ATEXIT_MAX
67258	{BC_BASE_MAX}	_SC_BC_BASE_MAX
67259	{BC_DIM_MAX}	_SC_BC_DIM_MAX
67260	{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
67261	{BC_STRING_MAX}	_SC_BC_STRING_MAX
67262	{CHILD_MAX}	_SC_CHILD_MAX
67263	Clock ticks/second	_SC_CLK_TCK
67264	{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
67265	{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
67266	{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
67267	{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
67268	{IOV_MAX}	_SC_IOV_MAX
67269	{LINE_MAX}	_SC_LINE_MAX
67270	{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
67271	{NGROUPS_MAX}	_SC_NGROUPS_MAX
67272	Initial size of <i>getgrgid_r()</i> and	_SC_GETGR_R_SIZE_MAX
67273	<i>getgrnam_r()</i> data buffers	
67274	Initial size of <i>getpwuid_r()</i> and	_SC_GETPW_R_SIZE_MAX
67275	<i>getpwnam_r()</i> data buffers	
67276	{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
67277	{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
67278	{OPEN_MAX}	_SC_OPEN_MAX
67279	{PAGE_SIZE}	_SC_PAGE_SIZE
67280	{PAGESIZE}	_SC_PAGESIZE
67281	{PTHREAD_DESTRUCTOR_ITERATIONS}	_SC_THREAD_DESTRUCTOR_ITERATIONS
67282	{PTHREAD_KEYS_MAX}	_SC_THREAD_KEYS_MAX
67283	{PTHREAD_STACK_MIN}	_SC_THREAD_STACK_MIN
67284	{PTHREAD_THREADS_MAX}	_SC_THREAD_THREADS_MAX
67285	{RE_DUP_MAX}	_SC_RE_DUP_MAX
67286	{RTSIG_MAX}	_SC_SIGRTMAX

	Variable	Value of Name
67287		
67288	{SEM_NSEMS_MAX}	_SC_SEM_NSEMS_MAX
67289	{SEM_VALUE_MAX}	_SC_SEM_VALUE_MAX
67290	{SIGQUEUE_MAX}	_SC_SIGQUEUE_MAX
67291	{STREAM_MAX}	_SC_STREAM_MAX
67292	{SYMLOOP_MAX}	_SC_SYMLOOP_MAX
67293	{TIMER_MAX}	_SC_TIMER_MAX
67294	{TTY_NAME_MAX}	_SC_TTY_NAME_MAX
67295	{TZNAME_MAX}	_SC_TZNAME_MAX
67296	_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
67297	_POSIX_BARRIERS	_SC_BARRIERS
67298	_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
67299	_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
67300	_POSIX_CPUTIME	_SC_CPUTIME
67301	_POSIX_FSYNC	_SC_FSYNC
67302	_POSIX_IPV6	_SC_IPV6
67303	_POSIX_JOB_CONTROL	_SC_JOB_CONTROL
67304	_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
67305	_POSIX_MEMLOCK	_SC_MEMLOCK
67306	_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
67307	_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
67308	_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
67309	_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
67310	_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
67311	_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
67312	_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
67313	_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
67314	_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
67315	_POSIX_REGEX	_SC_REGEX
67316	_POSIX_SAVED_IDS	_SC_SAVED_IDS
67317	_POSIX_SEMAPHORES	_SC_SEMAPHORES
67318	_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
67319	_POSIX_SHELL	_SC_SHELL
67320	_POSIX_SPAWN	_SC_SPAWN
67321	_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
67322	_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
67323	_POSIX_SS_REPL_MAX	_SC_SS_REPL_MAX
67324	_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
67325	_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
67326	_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
67327	_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
67328	_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
67329	_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
67330	_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
67331	_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
67332	_POSIX_THREAD_ROBUST_PRIO_INHERIT	_SC_THREAD_ROBUST_PRIO_INHERIT
67333	_POSIX_THREAD_ROBUST_PRIO_PROTECT	_SC_THREAD_ROBUST_PRIO_PROTECT
67334	_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
67335	_POSIX_THREAD_SPAWN	_SC_THREAD_SPAWN
67336	_POSIX_THREADS	_SC_THREADS
67337	_POSIX_TIMEOUTS	_SC_TIMEOUTS

	Variable	Value of Name
67338		
67339	_POSIX_TIMERS	_SC_TIMERS
67340	_POSIX_TRACE	_SC_TRACE
67341	_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
67342	_POSIX_TRACE_EVENT_NAME_MAX	_SC_TRACE_EVENT_NAME_MAX
67343	_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
67344	_POSIX_TRACE_LOG	_SC_TRACE_LOG
67345	_POSIX_TRACE_NAME_MAX	_SC_TRACE_NAME_MAX
67346	_POSIX_TRACE_SYS_MAX	_SC_TRACE_SYS_MAX
67347	_POSIX_TRACE_USER_EVENT_MAX	_SC_TRACE_USER_EVENT_MAX
67348	_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
67349	_POSIX_VERSION	_SC_VERSION
67350	_POSIX_V7_ILP32_OFF32	_SC_V7_ILP32_OFF32
67351	_POSIX_V7_ILP32_OFFBIG	_SC_V7_ILP32_OFFBIG
67352	_POSIX_V7_LP64_OFF64	_SC_V7_LP64_OFF64
67353	_POSIX_V7_LP64_OFFBIG	_SC_V7_LP64_OFFBIG
67354	OB _POSIX_V6_ILP32_OFF32	_SC_V6_ILP32_OFF32
67355	_POSIX_V6_ILP32_OFFBIG	_SC_V6_ILP32_OFFBIG
67356	_POSIX_V6_LP64_OFF64	_SC_V6_LP64_OFF64
67357	_POSIX_V6_LP64_OFFBIG	_SC_V6_LP64_OFFBIG
67358	_POSIX2_C_BIND	_SC_2_C_BIND
67359	_POSIX2_C_DEV	_SC_2_C_DEV
67360	_POSIX2_CHAR_TERM	_SC_2_CHAR_TERM
67361	_POSIX2_FORT_DEV	_SC_2_FORT_DEV
67362	_POSIX2_FORT_RUN	_SC_2_FORT_RUN
67363	_POSIX2_LOCALEDEF	_SC_2_LOCALEDEF
67364	_POSIX2_PBS	_SC_2_PBS
67365	_POSIX2_PBS_ACCOUNTING	_SC_2_PBS_ACCOUNTING
67366	_POSIX2_PBS_CHECKPOINT	_SC_2_PBS_CHECKPOINT
67367	_POSIX2_PBS_LOCATE	_SC_2_PBS_LOCATE
67368	_POSIX2_PBS_MESSAGE	_SC_2_PBS_MESSAGE
67369	_POSIX2_PBS_TRACK	_SC_2_PBS_TRACK
67370	_POSIX2_SW_DEV	_SC_2_SW_DEV
67371	_POSIX2_UPE	_SC_2_UPE
67372	_POSIX2_VERSION	_SC_2_VERSION
67373	_XOPEN_CRYPT	_SC_XOPEN_CRYPT
67374	_XOPEN_ENH_I18N	_SC_XOPEN_ENH_I18N
67375	_XOPEN_REALTIME	_SC_XOPEN_REALTIME
67376	_XOPEN_REALTIME_THREADS	_SC_XOPEN_REALTIME_THREADS
67377	_XOPEN_SHM	_SC_XOPEN_SHM
67378	_XOPEN_STREAMS	_SC_XOPEN_STREAMS
67379	_XOPEN_UNIX	_SC_XOPEN_UNIX
67380	_XOPEN_UUCP	_SC_XOPEN_UUCP
67381	_XOPEN_VERSION	_SC_XOPEN_VERSION

67382 RETURN VALUE

67383 If *name* is an invalid value, *sysconf()* shall return `-1` and set *errno* to indicate the error. If the
67384 variable corresponding to *name* is described in `<limits.h>` as a maximum or minimum value and
67385 the variable has no limit, *sysconf()* shall return `-1` without changing the value of *errno*. Note that
67386 indefinite limits do not imply infinite limits; see `<limits.h>`.

67387 Otherwise, *sysconf()* shall return the current variable value on the system. The value returned
67388 shall not be more restrictive than the corresponding value described to the application when it

67389 was compiled with the implementation's `<limits.h>` or `<unistd.h>`. The value shall not change
 67390 XSI during the lifetime of the calling process, except that `sysconf(_SC_OPEN_MAX)` may return
 67391 different values before and after a call to `setrlimit()` which changes the `RLIMIT_NOFILE` soft
 67392 limit.

67393 If the variable corresponding to *name* is dependent on an unsupported option, the results are
 67394 unspecified.

67395 ERRORS

67396 The `sysconf()` function shall fail if:

67397 [EINVAL] The value of the *name* argument is invalid.

67398 EXAMPLES

67399 None.

67400 APPLICATION USAGE

67401 As `-1` is a permissible return value in a successful situation, an application wishing to check for
 67402 error situations should set `errno` to 0, then call `sysconf()`, and, if it returns `-1`, check to see if `errno`
 67403 is non-zero.

67404 Application developers should check whether an option, such as `_POSIX_TRACE`, is supported
 67405 prior to obtaining and using values for related variables, such as `_POSIX_TRACE_NAME_MAX`.

67406 RATIONALE

67407 This functionality was added in response to requirements of application developers and of
 67408 system vendors who deal with many international system configurations. It is closely related to
 67409 `pathconf()` and `fpathconf()`.

67410 Although a conforming application can run on all systems by never demanding more resources
 67411 than the minimum values published in this volume of POSIX.1-2017, it is useful for that
 67412 application to be able to use the actual value for the quantity of a resource available on any
 67413 given system. To do this, the application makes use of the value of a symbolic constant in
 67414 `<limits.h>` or `<unistd.h>`.

67415 However, once compiled, the application must still be able to cope if the amount of resource
 67416 available is increased. To that end, an application may need a means of determining the quantity
 67417 of a resource, or the presence of an option, at execution time.

67418 Two examples are offered:

- 67419 1. Applications may wish to act differently on systems with or without job control.
 67420 Applications vendors who wish to distribute only a single binary package to all instances
 67421 of a computer architecture would be forced to assume job control is never available if it
 67422 were to rely solely on the `<unistd.h>` value published in this volume of POSIX.1-2017.
- 67423 2. International applications vendors occasionally require knowledge of the number of clock
 67424 ticks per second. Without these facilities, they would be required to either distribute their
 67425 applications partially in source form or to have 50 Hz and 60 Hz versions for the various
 67426 countries in which they operate.

67427 It is the knowledge that many applications are actually distributed widely in executable form
 67428 that leads to this facility. If limited to the most restrictive values in the headers, such applications
 67429 would have to be prepared to accept the most limited environments offered by the smallest
 67430 microcomputers. Although this is entirely portable, there was a consensus that they should be
 67431 able to take advantage of the facilities offered by large systems, without the restrictions
 67432 associated with source and object distributions.

67433 During the discussions of this feature, it was pointed out that it is almost always possible for an

67434 application to discern what a value might be at runtime by suitably testing the various functions
 67435 themselves. And, in any event, it could always be written to adequately deal with error returns
 67436 from the various functions. In the end, it was felt that this imposed an unreasonable level of
 67437 complication and sophistication on the application developer.

67438 This runtime facility is not meant to provide ever-changing values that applications have to
 67439 check multiple times. The values are seen as changing no more frequently than once per system
 67440 initialization, such as by a system administrator or operator with an automatic configuration
 67441 program. This volume of POSIX.1-2017 specifies that they shall not change within the lifetime of
 67442 the process.

67443 Some values apply to the system overall and others vary at the file system or directory level. The
 67444 latter are described in *fpathconf()*.

67445 Note that all values returned must be expressible as integers. String values were considered, but
 67446 the additional flexibility of this approach was rejected due to its added complexity of
 67447 implementation and use.

67448 Some values, such as {PATH_MAX}, are sometimes so large that they must not be used to, say,
 67449 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic
 67450 constant is not even defined in this case.

67451 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is
 67452 infinite, returning an error indicating that some other resource limit has been reached is
 67453 conforming behavior.

67454 **FUTURE DIRECTIONS**

67455 None.

67456 **SEE ALSO**

67457 *confstr()*, *fpathconf()*

67458 XBD [<limits.h>](#), [<unistd.h>](#)

67459 XCU *getconf*

67460 **CHANGE HISTORY**

67461 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67462 **Issue 5**

67463 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 67464 Threads Extension.

67465 The `_XBS_` variables and name values are added to the table of system variables in the
 67466 DESCRIPTION. These are all marked EX.

67467 **Issue 6**

67468 The symbol CLK_TCK is obsolescent and removed. It is replaced with the phrase “clock ticks
 67469 per second”.

67470 The symbol {PASS_MAX} is removed.

67471 The following changes were made to align with the IEEE P1003.1a draft standard:

67472 Table entries are added for the following variables: `_SC_REGEX`, `_SC_SHELL`,
 67473 `_SC_REGEX_VERSION`, `_SC_SYMLOOP_MAX`.

67474 The following *sysconf()* variables and their associated names are added for alignment with
 67475 IEEE Std 1003.1d-1999:

67476 _POSIX_ADVISORY_INFO
 67477 _POSIX_CPUTIME
 67478 _POSIX_SPAWN
 67479 _POSIX_SPORADIC_SERVER
 67480 _POSIX_THREAD_CPUTIME
 67481 _POSIX_THREAD_SPORADIC_SERVER
 67482 _POSIX_TIMEOUTS

67483 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

67484 A statement expressing the dependency of support for some system variables on
 67485 implementation options is added.

67486 The following system variables are added:

67487 _POSIX_BARRIERS
 67488 _POSIX_CLOCK_SELECTION
 67489 _POSIX_MONOTONIC_CLOCK
 67490 _POSIX_READER_WRITER_LOCKS
 67491 _POSIX_SPIN_LOCKS
 67492 _POSIX_TYPED_MEMORY_OBJECTS

67493 The following system variables are added for alignment with IEEE Std 1003.2d-1994:

67494 _POSIX2_PBS
 67495 _POSIX2_PBS_ACCOUNTING
 67496 _POSIX2_PBS_LOCATE
 67497 _POSIX2_PBS_MESSAGE
 67498 _POSIX2_PBS_TRACK

67499 The following *sysconf()* variables and their associated names are added for alignment with
 67500 IEEE Std 1003.1q-2000:

67501 _POSIX_TRACE
 67502 _POSIX_TRACE_EVENT_FILTER
 67503 _POSIX_TRACE_INHERIT
 67504 _POSIX_TRACE_LOG

67505 The macros associated with the *c89* programming models are marked LEGACY, and new
 67506 equivalent macros associated with *c99* are introduced.

67507 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the
 67508 DESCRIPTION to denote that the `_PC*` and `_SC*` symbols are now required to be supported. A
 67509 corresponding change has been made in the Base Definitions volume of POSIX.1-2017. The
 67510 deletion in the second paragraph removes some duplicated text. Additional symbols that were
 67511 erroneously omitted from this reference page have been added.

67512 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the
 67513 RETURN VALUE section that the value returned for *sysconf*(`_SC_OPEN_MAX`) may change if a
 67514 call to *setrlimit*() adjusts the `RLIMIT_NOFILE` soft limit.

67515 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/134 is applied, updating the
 67516 DESCRIPTION to remove an erroneous entry for `_POSIX_SYMLINK_MAX`. This corrects an
 67517 error in IEEE Std 1003.1-2001/Cor 1-2002.

67518 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/135 is applied, removing

67519 _posix_file_locking, _posix_multi_process, _posix2_c_version, and
67520 _xopen_xcu_version (and their associated _SC_* variables) from the DESCRIPTION and
67521 APPLICATION USAGE sections.

67522 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/136 is applied, adding the following
67523 constants (and their associated _SC_* variables) to the DESCRIPTION:

67524 _posix_ss_repl_max
67525 _posix_trace_event_name_max
67526 _posix_trace_name_max
67527 _posix_trace_sys_max
67528 _posix_trace_user_event_max

67529 The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables
67530 are dependent on unsupported options, the results are unspecified.

67531 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/137 is applied, removing
67532 _regex_version and _SC_REGEX_VERSION.

67533 **Issue 7**

67534 Austin Group Interpretation 1003.1-2001 #160 is applied.

67535 SD5-XSH-ERN-166 is applied, changing “Maximum size” to “Initial size” for the “Maximum
67536 size of ...” entries in the table in the DESCRIPTION.

67537 The variables for the supported programming environments are updated to be V7 and the
67538 LEGACY variables are removed.

67539 The following constants are added:

67540 _posix_thread_robust_prio_inherit
67541 _posix_thread_robust_prio_protect

67542 The _XOPEN_UUCP variable and its associated _SC_XOPEN_UUCP value is added to the table
67543 of system variables.

67544 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0364 [752] is applied.

67545 **NAME**

67546 syslog ‡log a message

67547 **SYNOPSIS**

```
67548 XSI #include <syslog.h>
67549 void syslog(int priority, const char *message, ... /* argument */);
```

67550 **DESCRIPTION**

67551 Refer to *closelog()*.

67552 **NAME**

67553 system ‡issue a command

67554 **SYNOPSIS**

67555 #include <stdlib.h>

67556 int system(const char *command);

67557 **DESCRIPTION**

67558 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67559 conflict between the requirements described here and the ISO C standard is unintentional. This
 67560 volume of POSIX.1-2017 defers to the ISO C standard.

67561 If *command* is a null pointer, the *system()* function shall determine whether the host environment
 67562 has a command processor. If *command* is not a null pointer, the *system()* function shall pass the
 67563 string pointed to by *command* to that command processor to be executed in an implementation-
 67564 defined manner; this might then cause the program calling *system()* to behave in a non-
 67565 conforming manner or to terminate.

67566 CX The *system()* function shall behave as if a child process were created using *fork()*, and the child
 67567 process invoked the *sh* utility using *execl()* as follows:

67568 `execl(<shell path>, "sh", "-c", command, (char *)0);`

67569 where <shell path> is an unspecified pathname for the *sh* utility. It is unspecified whether the
 67570 handlers registered with *pthread_atfork()* are called as part of the creation of the child process.

67571 The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the
 67572 SIGCHLD signal, while waiting for the command to terminate. If this might cause the
 67573 application to miss a signal that would have killed it, then the application should examine the
 67574 return value from *system()* and take whatever action is appropriate to the application if the
 67575 command terminated due to receipt of a signal.

67576 The *system()* function shall not affect the termination status of any child of the calling processes
 67577 other than the process or processes it itself creates.

67578 The *system()* function shall not return until the child process has terminated.

67579 The *system()* function need not be thread-safe.

67580 **RETURN VALUE**

67581 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor
 67582 CX is available, or zero if none is available. The *system()* function shall always return non-zero
 67583 when *command* is NULL.

67584 CX If *command* is not a null pointer, *system()* shall return the termination status of the command
 67585 language interpreter in the format specified by *waitpid()*. The termination status shall be as
 67586 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents
 67587 the command language interpreter from executing after the child process is created, the return
 67588 value from *system()* shall be as if the command language interpreter had terminated using
 67589 *exit(127)* or *_exit(127)*. If a child process cannot be created, or if the termination status for the
 67590 command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to
 67591 indicate the error.

67592 **ERRORS**

67593 CX The *system()* function may set *errno* values as described by *fork()*.

67594 In addition, *system()* may fail if:

67595 CX [ECHILD] The status of the child process created by *system()* is no longer available.

67596 EXAMPLES

67597 None.

67598 APPLICATION USAGE

67599 If the return value of *system()* is not -1 , its value can be decoded through the use of the macros
67600 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.

67601 Note that, while *system()* must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting
67602 for the child to terminate, the handling of signals in the executed command is as specified by
67603 *fork()* and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system()* is
67604 called, then the child is started with SIGINT handling set to SIG_DFL.

67605 Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two
67606 processes reading from the same terminal, for example) when the executed command ignores or
67607 catches one of the signals. It is also usually the correct action when the user has given a
67608 command to the application to be executed synchronously (as in the '!' command in many
67609 interactive applications). In either case, the signal should be delivered only to the child process,
67610 not to the application itself. There is one situation where ignoring the signals might have less
67611 than the desired effect. This is when the application uses *system()* to perform some task invisible
67612 to the user. If the user typed the interrupt character (" $\text{^\text{C}}$ ", for example) while *system()* is being
67613 used in this way, one would expect the application to be killed, but only the executed command
67614 is killed. Applications that use *system()* in this way should carefully check the return status from
67615 *system()* to see if the executed command was successful, and should take appropriate action
67616 when the command fails.

67617 Blocking SIGCHLD while waiting for the child to terminate prevents the application from
67618 catching the signal and obtaining status from *system()*'s child process before *system()* can get the
67619 status itself.

67620 The context in which the utility is ultimately executed may differ from that in which *system()*
67621 was called. For example, file descriptors that have the FD_CLOEXEC flag set are closed, and the
67622 process ID and parent process ID are different. Also, if the executed utility changes its
67623 environment variables or its current working directory, that change is not reflected in the caller's
67624 context.

67625 There is no defined way for an application to find the specific path for the shell. However,
67626 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.

67627 Using the *system()* function in more than one thread in a process or when the SIGCHLD signal is
67628 being manipulated by more than one thread in a process may produce unexpected results.

67629 RATIONALE

67630 The *system()* function should not be used by programs that have set user (or group) ID
67631 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used
67632 instead. This prevents any unforeseen manipulation of the environment of the user that could
67633 cause execution of commands not anticipated by the calling program.

67634 There are three levels of specification for the *system()* function. The ISO C standard gives the
67635 most basic. It requires that the function exists, and defines a way for an application to query
67636 whether a command language interpreter exists. It says nothing about the command language
67637 or the environment in which the command is interpreted.

67638 POSIX.1-2017 places additional restrictions on *system()*. It requires that if there is a command
67639 language interpreter, the environment must be as specified by *fork()* and *exec*. This ensures, for

67640 example, that *close-on-exec* works, that file locks are not inherited, and that the process ID is
 67641 different. It also specifies the return value from *system()* when the command line can be run,
 67642 thus giving the application some information about the command's completion status.

67643 Finally, POSIX.1-2017 requires the command to be interpreted as in the shell command language
 67644 defined in the Shell and Utilities volume of POSIX.1-2017.

67645 Note that, *system(NULL)* is required to return non-zero, indicating that there is a command
 67646 language interpreter. At first glance, this would seem to conflict with the ISO C standard which
 67647 allows *system(NULL)* to return zero. There is no conflict, however. A system must have a
 67648 command language interpreter, and is non-conforming if none is present. It is therefore
 67649 permissible for the *system()* function on such a system to implement the behavior specified by
 67650 the ISO C standard as long as it is understood that the implementation does not conform to
 67651 POSIX.1-2017 if *system(NULL)* returns zero.

67652 It was explicitly decided that when *command* is NULL, *system()* should not be required to check
 67653 to make sure that the command language interpreter actually exists with the correct mode, that
 67654 there are enough processes to execute it, and so on. The call *system(NULL)* could, theoretically,
 67655 check for such problems as too many existing child processes, and return zero. However, it
 67656 would be inappropriate to return zero due to such a (presumably) transient condition. If some
 67657 condition exists that is not under the control of this application and that would cause any
 67658 *system()* call to fail, that system has been rendered non-conforming.

67659 Early drafts required, or allowed, *system()* to return with *errno* set to [EINTR] if it was
 67660 interrupted with a signal. This error return was removed, and a requirement that *system()* not
 67661 return until the child has terminated was added. This means that if a *waitpid()* call in *system()*
 67662 exits with *errno* set to [EINTR], *system()* must reissue the *waitpid()*. This change was made for
 67663 two reasons:

- 67664 1. There is no way for an application to clean up if *system()* returns [EINTR], short of calling
 67665 *wait()*, and that could have the undesirable effect of returning the status of children other
 67666 than the one started by *system()*.
- 67667 2. While it might require a change in some historical implementations, those
 67668 implementations already have to be changed because they use *wait()* instead of *waitpid()*.

67669 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a
 67670 successful *system()* call returns.

67671 To conform to POSIX.1-2017, *system()* must use *waitpid()*, or some similar function, instead of
 67672 *wait()*.

67673 The following code sample illustrates how *system()* might be implemented on an
 67674 implementation conforming to POSIX.1-2017.

```
67675 #include <signal.h>
67676 int system(const char *cmd)
67677 {
67678     int stat;
67679     pid_t pid;
67680     struct sigaction sa, savintr, savequit;
67681     sigset_t saveblock;
67682     if (cmd == NULL)
67683         return(1);
67684     sa.sa_handler = SIG_IGN;
67685     sigemptyset(&sa.sa_mask);
67686     sa.sa_flags = 0;
```

```

67687     sigemptyset(&saveintr.sa_mask);
67688     sigemptyset(&savequit.sa_mask);
67689     sigaction(SIGINT, &sa, &saveintr);
67690     sigaction(SIGQUIT, &sa, &savequit);
67691     sigaddset(&sa.sa_mask, SIGCHLD);
67692     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
67693     if ((pid = fork()) == 0) {
67694         sigaction(SIGINT, &saveintr, (struct sigaction *)0);
67695         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
67696         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
67697         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
67698         _exit(127);
67699     }
67700     if (pid == -1) {
67701         stat = -1; /* errno comes from fork() */
67702     } else {
67703         while (waitpid(pid, &stat, 0) == -1) {
67704             if (errno != EINTR){
67705                 stat = -1;
67706                 break;
67707             }
67708         }
67709     }
67710     sigaction(SIGINT, &saveintr, (struct sigaction *)0);
67711     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
67712     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
67713     return(stat);
67714 }

```

67715 Note that, while a particular implementation of *system()* (such as the one above) can assume a
67716 particular path for the shell, such a path is not necessarily valid on another system. The above
67717 example is not portable, and is not intended to be.

67718 Note also that the above example implementation is not thread-safe. Implementations can
67719 provide a thread-safe *system()* function, but doing so involves complications such as how to
67720 restore the signal dispositions for SIGINT and SIGQUIT correctly if there are overlapping calls,
67721 and how to deal with cancellation. The example above would not restore the signal dispositions
67722 and would leak a process ID if cancelled. This does not matter for a non-thread-safe
67723 implementation since canceling a non-thread-safe function results in undefined behavior (see
67724 [Section 2.9.5.2](#), on page 518). To avoid leaking a process ID, a thread-safe implementation would
67725 need to terminate the child process when acting on a cancellation.

67726 One reviewer suggested that an implementation of *system()* might want to use an environment
67727 variable such as *SHELL* to determine which command interpreter to use. The supposed
67728 implementation would use the default command interpreter if the one specified by the
67729 environment variable was not available. This would allow a user, when using an application that
67730 prompts for command lines to be processed using *system()*, to specify a different command
67731 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not
67732 follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-2017, then
67733 changing *SHELL* would render *system()* non-conforming. This would affect applications that
67734 expected the specified behavior from *system()*, and since the Shell and Utilities volume of
67735 POSIX.1-2017 does not mention that *SHELL* affects *system()*, the application would not know
67736 that it needed to unset *SHELL*.

67737 **FUTURE DIRECTIONS**

67738 None.

67739 **SEE ALSO**67740 Section 2.9.5.2 (on page 518), *exec*, *pipe()*, *pthread_atfork()*, *wait()*67741 XBD [<limits.h>](#), [<signal.h>](#), [<stdlib.h>](#), [<sys/wait.h>](#)67742 XCU *sh*67743 **CHANGE HISTORY**

67744 First released in Issue 1. Derived from Issue 1 of the SVID.

67745 **Issue 6**

67746 Extensions beyond the ISO C standard are marked.

67747 **Issue 7**67748 Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this
67749 function and treatment of *at_fork()* handlers.

67750 Austin Group Interpretation 1003.1-2001 #156 is applied.

67751 SD5-XSH-ERN-30 is applied.

67752 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0365 [627] is applied.

67753 **NAME**

67754 tan, tanf, tanl ‡tangent function

67755 **SYNOPSIS**

```
67756 #include <math.h>
67757 double tan(double x);
67758 float tanf(float x);
67759 long double tanl(long double x);
```

67760 **DESCRIPTION**

67761 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67762 conflict between the requirements described here and the ISO C standard is unintentional. This
 67763 volume of POSIX.1-2017 defers to the ISO C standard.

67764 These functions shall compute the tangent of their argument x , measured in radians.

67765 An application wishing to check for error situations should set *errno* to zero and call
 67766 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 67767 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 67768 zero, an error has occurred.

67769 **RETURN VALUE**

67770 Upon successful completion, these functions shall return the tangent of x .

67771 MXX If the correct value would cause underflow, and is not representable, a range error may occur,
 67772 MXX and *tan()*, *tanf()*, and *tanl()* shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an
 67773 implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and
 67774 LDBL_MIN, respectively.

67775 MX If x is NaN, a NaN shall be returned.

67776 If x is ± 0 , x shall be returned.

67777 If x is subnormal, a range error may occur

67778 MXX and x should be returned.

67779 MX If x is not returned, *tan()*, *tanf()*, and *tanl()* shall return an implementation-defined value no
 67780 greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

67781 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 67782 defined value shall be returned.

67783 MXX If the correct value would cause underflow, and is representable, a range error may occur and
 67784 the correct value shall be returned.

67785 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*
 67786 shall return $\pm\text{HUGE_VAL}$, $\pm\text{HUGE_VALF}$, and $\pm\text{HUGE_VALL}$, respectively, with the same sign
 67787 as the correct value of the function.

67788 **ERRORS**

67789 These functions shall fail if:

67790 MX **Domain Error** The value of x is $\pm\text{Inf}$.

67791 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 67792 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 67793 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 67794 shall be raised.

67795 XSI **Range Error** The result overflows
 67796 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 67797 then *errno* shall be set to [ERANGE]. If the integer expression
 67798 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 67799 floating-point exception shall be raised.

67800 These functions may fail if:

67801 MX **Range Error** The result underflows, or the value of *x* is subnormal.
 67802 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 67803 then *errno* shall be set to [ERANGE]. If the integer expression
 67804 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 67805 floating-point exception shall be raised.

67806 EXAMPLES

67807 Taking the Tangent of a 45-Degree Angle

```
67808 #include <math.h>
67809 ...
67810 double radians = 45.0 * M_PI / 180;
67811 double result;
67812 ...
67813 result = tan (radians);
```

67814 APPLICATION USAGE

67815 There are no known floating-point representations such that for a normal argument, $\tan(x)$ is
 67816 either overflow or underflow.

67817 These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from
 67818 0.0.

67819 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 67820 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

67821 RATIONALE

67822 None.

67823 FUTURE DIRECTIONS

67824 None.

67825 SEE ALSO

67826 [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

67827 XBD [Section 4.20](#) (on page 117), [<math.h>](#)

67828 CHANGE HISTORY

67829 First released in Issue 1. Derived from Issue 1 of the SVID.

67830 Issue 5

67831 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
 67832 in previous issues.

67833 Issue 6

67834 The [tanf\(\)](#) and [tanl\(\)](#) functions are added for alignment with the ISO/IEC 9899:1999 standard.

67835 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 67836 revised to align with the ISO/IEC 9899:1999 standard.

67837 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
67838 marked.

67839 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last
67840 paragraph in the RETURN VALUE section.

67841 **Issue 7**

67842 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0635 [68], XSH/TC1-2008/0636 [68],
67843 and XSH/TC1-2008/0637 [68] are applied.

67844 **NAME**

67845 tanh, tanhf, tanhl ‡hyperbolic tangent functions

67846 **SYNOPSIS**

```
67847 #include <math.h>
67848 double tanh(double x);
67849 float tanhf(float x);
67850 long double tanhl(long double x);
```

67851 **DESCRIPTION**

67852 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67853 conflict between the requirements described here and the ISO C standard is unintentional. This
 67854 volume of POSIX.1-2017 defers to the ISO C standard.

67855 These functions shall compute the hyperbolic tangent of their argument x .

67856 An application wishing to check for error situations should set *errno* to zero and call
 67857 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 67858 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 67859 zero, an error has occurred.

67860 **RETURN VALUE**

67861 Upon successful completion, these functions shall return the hyperbolic tangent of x .

67862 MX If x is NaN, a NaN shall be returned.

67863 If x is ± 0 , x shall be returned.

67864 If x is $\pm\text{Inf}$, ± 1 shall be returned.

67865 If x is subnormal, a range error may occur
 67866 MXX and x should be returned.

67867 MX If x is not returned, *tanh()*, *tanhf()*, and *tanhl()* shall return an implementation-defined value no
 67868 greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

67869 **ERRORS**

67870 These functions may fail if:

67871 MX **Range Error** The value of x is subnormal.

67872 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 67873 then *errno* shall be set to [ERANGE]. If the integer expression
 67874 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 67875 floating-point exception shall be raised.

67876 **EXAMPLES**

67877 None.

67878 **APPLICATION USAGE**

67879 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 67880 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

67881 **RATIONALE**

67882 None.

67883 **FUTURE DIRECTIONS**

67884 None.

67885 **SEE ALSO**67886 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*67887 XBD Section 4.20 (on page 117), **<math.h>**67888 **CHANGE HISTORY**

67889 First released in Issue 1. Derived from Issue 1 of the SVID.

67890 **Issue 5**67891 The DESCRIPTION is updated to indicate how an application should check for an error. This
67892 text was previously published in the APPLICATION USAGE section.67893 **Issue 6**67894 The *tanhf()* and *tanhfll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.67895 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
67896 revised to align with the ISO/IEC 9899:1999 standard.67897 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
67898 marked.67899 **Issue 7**

67900 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0638 [68] is applied.

67901 **NAME**

67902 tanl ‡tangent function

67903 **SYNOPSIS**

67904 #include <math.h>

67905 long double tanl(long double x);

67906 **DESCRIPTION**

67907 Refer to *tan()*.

67908 **NAME**

67909 tcdrain ¶wait for transmission of output

67910 **SYNOPSIS**

67911 #include <termios.h>

67912 int tcdrain(int *fildev*);67913 **DESCRIPTION**67914 The *tcdrain()* function shall block until all output written to the object referred to by *fildev* is
67915 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.67916 Any attempts to use *tcdrain()* from a process which is a member of a background process group
67917 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a
67918 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
67919 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.67920 **RETURN VALUE**67921 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
67922 indicate the error.67923 **ERRORS**67924 The *tcdrain()* function shall fail if:67925 [EBADF] The *fildev* argument is not a valid file descriptor.67926 [EINTR] A signal interrupted *tcdrain()*.67927 [EIO] The process group of the writing process is orphaned, the calling thread is not
67928 blocking SIGTTOU, and the process is not ignoring SIGTTOU.67929 [ENOTTY] The file associated with *fildev* is not a terminal.67930 **EXAMPLES**

67931 None.

67932 **APPLICATION USAGE**

67933 None.

67934 **RATIONALE**

67935 None.

67936 **FUTURE DIRECTIONS**

67937 None.

67938 **SEE ALSO**67939 [tcflush\(\)](#)67940 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)67941 **CHANGE HISTORY**

67942 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

67943 **Issue 6**

67944 The following new requirements on POSIX implementations derive from alignment with the
67945 Single UNIX Specification:

67946 In the DESCRIPTION, the final paragraph is no longer conditional on
67947 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.

67948 The [EIO] error is added.

67949 **Issue 7**

67950 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0639 [79], XSH/TC1-2008/0640 [79],
67951 and XSH/TC1-2008/0641 [79] are applied.

67952 **NAME**

67953 tcflow — suspend or restart the transmission or reception of data

67954 **SYNOPSIS**

```
67955 #include <termios.h>
67956 int tcflow(int fildes, int action);
```

67957 **DESCRIPTION**

67958 The *tcflow()* function shall suspend or restart transmission or reception of data on the object
 67959 referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file
 67960 descriptor associated with a terminal.

67961 If *action* is TCOOFF, output shall be suspended.

67962 If *action* is TCOON, suspended output shall be restarted.

67963 If *action* is TCIOFF and *fildes* refers to a terminal device, the system shall transmit a STOP
 67964 character, which is intended to cause the terminal device to stop transmitting data to the
 67965 system. If *fildes* is associated with a pseudo-terminal, the STOP character need not be
 67966 transmitted.

67967 If *action* is TCION and *fildes* refers to a terminal device, the system shall transmit a START
 67968 character, which is intended to cause the terminal device to start transmitting data to the
 67969 system. If *fildes* is associated with a pseudo-terminal, the START character need not be
 67970 transmitted.

67971 The default on the opening of a terminal file is that neither its input nor its output are
 67972 suspended.

67973 Attempts to use *tcflow()* from a process which is a member of a background process group on a
 67974 *fildes* associated with its controlling terminal, shall cause the process group to be sent a
 67975 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
 67976 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

67977 **RETURN VALUE**

67978 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 67979 indicate the error.

67980 **ERRORS**

67981 The *tcflow()* function shall fail if:

67982 [EBADF] The *fildes* argument is not a valid file descriptor.

67983 [EINVAL] The *action* argument is not a supported value.

67984 [EIO] The process group of the writing process is orphaned, the calling thread is not
 67985 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

67986 [ENOTTY] The file associated with *fildes* is not a terminal.

67987 **EXAMPLES**

67988 None.

67989 **APPLICATION USAGE**

67990 None.

67991 **RATIONALE**

67992 None.

67993 **FUTURE DIRECTIONS**

67994 None.

67995 **SEE ALSO**67996 [tcsendbreak\(\)](#)67997 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)67998 **CHANGE HISTORY**

67999 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68000 **Issue 6**68001 The following new requirements on POSIX implementations derive from alignment with the
68002 Single UNIX Specification:

68003 The [EIO] error is added.

68004 **Issue 7**68005 SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and
68006 STOP characters.68007 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0642 [79], XSH/TC1-2008/0643 [79],
68008 and XSH/TC1-2008/0644 [79] are applied.

68009 **NAME**

68010 tcflush — flush non-transmitted output data, non-read input data, or both

68011 **SYNOPSIS**

68012 #include <termios.h>

68013 int tcflush(int *fildev*, int *queue_selector*);68014 **DESCRIPTION**68015 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*
68016 (an open file descriptor associated with a terminal) but not transmitted, or data received but not
68017 read, depending on the value of *queue_selector*:68018 If *queue_selector* is TCIFLUSH, it shall flush data received but not read.68019 If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.68020 If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data
68021 written but not transmitted.68022 Attempts to use *tcflush()* from a process which is a member of a background process group on a
68023 *fildev* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU
68024 signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU
68025 signals, the process shall be allowed to perform the operation, and no signal is sent.68026 **RETURN VALUE**68027 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
68028 indicate the error.68029 **ERRORS**68030 The *tcflush()* function shall fail if:68031 [EBADF] The *fildev* argument is not a valid file descriptor.68032 [EINVAL] The *queue_selector* argument is not a supported value.68033 [EIO] The process group of the writing process is orphaned, the calling thread is not
68034 blocking SIGTTOU, and the process is not ignoring SIGTTOU.68035 [ENOTTY] The file associated with *fildev* is not a terminal.68036 **EXAMPLES**

68037 None.

68038 **APPLICATION USAGE**

68039 None.

68040 **RATIONALE**

68041 None.

68042 **FUTURE DIRECTIONS**

68043 None.

68044 **SEE ALSO**68045 [tcdrain\(\)](#)68046 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)68047 **CHANGE HISTORY**

68048 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68049 **Issue 6**

68050 The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE
68051 sections, references to *tcflow()* are replaced with *tcflush()*.

68052 The following new requirements on POSIX implementations derive from alignment with the
68053 Single UNIX Specification:

68054 In the DESCRIPTION, the final paragraph is no longer conditional on
68055 `_POSIX_JOB_CONTROL`. This is a FIPS requirement.

68056 The [EIO] error is added.

68057 **Issue 7**

68058 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0645 [79], XSH/TC1-2008/0646 [79],
68059 and XSH/TC1-2008/0647 [79] are applied.

68060 **NAME**

68061 tcgetattr ¶get the parameters associated with the terminal

68062 **SYNOPSIS**

68063 #include <termios.h>

68064 int tcgetattr(int *fildev*, struct termios **termios_p*);68065 **DESCRIPTION**

68066 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev*
 68067 and store them in the **termios** structure referenced by *termios_p*. The *fildev* argument is an open
 68068 file descriptor associated with a terminal.

68069 The *termios_p* argument is a pointer to a **termios** structure.

68070 The *tcgetattr()* operation is allowed from any process.

68071 If the terminal device supports different input and output baud rates, the baud rates stored in
 68072 the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are
 68073 equal. If differing baud rates are not supported, the rate returned as the output baud rate shall
 68074 be the actual baud rate. If the terminal device does not support split baud rates, the input baud
 68075 rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).

68076 **RETURN VALUE**

68077 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 68078 indicate the error.

68079 **ERRORS**

68080 The *tcgetattr()* function shall fail if:

68081 [EBADF] The *fildev* argument is not a valid file descriptor.

68082 [ENOTTY] The file associated with *fildev* is not a terminal.

68083 **EXAMPLES**

68084 None.

68085 **APPLICATION USAGE**

68086 None.

68087 **RATIONALE**

68088 Care must be taken when changing the terminal attributes. Applications should always do a
 68089 *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()*, changing only
 68090 the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the
 68091 terminal state whenever it is done with the terminal. This is necessary because terminal
 68092 attributes apply to the underlying port and not to each individual open instance; that is, all
 68093 processes that have used the terminal see the latest attribute changes.

68094 A program that uses these functions should be written to catch all signals and take other
 68095 appropriate actions to ensure that when the program terminates, whether planned or not, the
 68096 terminal device's state is restored to its original state.

68097 Existing practice dealing with error returns when only part of a request can be honored is based
 68098 on calls to the *ioctl()* function. In historical BSD and System V implementations, the
 68099 corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if
 68100 some of the requested changes could not be made. Many existing applications assume this
 68101 behavior and would no longer work correctly if the return value were changed from zero to -1
 68102 in this case.

68103 Note that either specification has a problem. When zero is returned, it implies everything

68104 succeeded even if some of the changes were not made. When -1 is returned, it implies
68105 everything failed even though some of the changes were made.

68106 Applications that need all of the requested changes made to work properly should follow
68107 *tcsetattr()* with a call to *tcgetattr()* and compare the appropriate field values.

68108 **FUTURE DIRECTIONS**

68109 None.

68110 **SEE ALSO**

68111 [tcsetattr\(\)](#)

68112 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)

68113 **CHANGE HISTORY**

68114 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68115 **Issue 6**

68116 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.
68117 Previously, the number zero was also allowed but was obsolescent.

68118 **NAME**

68119 tcgetpgrp — get the foreground process group ID

68120 **SYNOPSIS**

68121 #include <unistd.h>

68122 pid_t tcgetpgrp(int *fildev*);68123 **DESCRIPTION**68124 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process
68125 group associated with the terminal.68126 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does
68127 not match the process group ID of any existing process group.68128 The *tcgetpgrp()* function is allowed from a process that is a member of a background process
68129 group; however, the information may be subsequently changed by a process that is a member of
68130 a foreground process group.68131 **RETURN VALUE**68132 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the
68133 foreground process associated with the terminal. Otherwise, -1 shall be returned and *errno* set to
68134 indicate the error.68135 **ERRORS**68136 The *tcgetpgrp()* function shall fail if:68137 [EBADF] The *fildev* argument is not a valid file descriptor.68138 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
68139 controlling terminal.68140 **EXAMPLES**

68141 None.

68142 **APPLICATION USAGE**

68143 None.

68144 **RATIONALE**

68145 None.

68146 **FUTURE DIRECTIONS**

68147 None.

68148 **SEE ALSO**68149 [setsid\(\)](#), [setpgid\(\)](#), [tcsetpgrp\(\)](#)68150 XBD [<sys/types.h>](#), [<unistd.h>](#)68151 **CHANGE HISTORY**

68152 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68153 **Issue 6**68154 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.68155 The following new requirements on POSIX implementations derive from alignment with the
68156 Single UNIX Specification:68157 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
68158 required for conforming implementations of previous POSIX specifications, it was not
68159 required for UNIX applications.

68160
68161

In the DESCRIPTION, text previously conditional on support for `_POSIX_JOB_CONTROL` is now mandatory. This is a FIPS requirement.

68162 **NAME**

68163 tcgetsid — get the process group ID for the session leader for the controlling terminal

68164 **SYNOPSIS**

```
68165 #include <termios.h>
68166 pid_t tcgetsid(int fildev);
```

68167 **DESCRIPTION**

68168 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal
68169 specified by *fildev* is the controlling terminal.

68170 **RETURN VALUE**

68171 Upon successful completion, *tcgetsid()* shall return the process group ID of the session
68172 associated with the terminal. Otherwise, a value of -1 shall be returned and *errno* set to indicate
68173 the error.

68174 **ERRORS**

68175 The *tcgetsid()* function shall fail if:

68176	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor.
68177	[ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal.

68179 **EXAMPLES**

68180 None.

68181 **APPLICATION USAGE**

68182 None.

68183 **RATIONALE**

68184 None.

68185 **FUTURE DIRECTIONS**

68186 None.

68187 **SEE ALSO**

68188 XBD [<termios.h>](#)

68189 **CHANGE HISTORY**

68190 First released in Issue 4, Version 2.

68191 **Issue 5**

68192 Moved from X/OPEN UNIX extension to BASE.

68193 The [EACCES] error has been removed from the list of mandatory errors, and the description of
68194 [ENOTTY] has been reworded.

68195 **Issue 7**

68196 SD5-XSH-ERN-180 is applied, clarifying the RETURN VALUE section.

68197 The *tcgetsid()* function is moved from the XSI option to the Base.

68198 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0648 [421] is applied.

68199 **NAME**

68200 tcsendbreak — send a break for a specific duration

68201 **SYNOPSIS**

68202 #include <termios.h>

68203 int tcsendbreak(int *fildev*, int *duration*);68204 **DESCRIPTION**

68205 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause
68206 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it
68207 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5
68208 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period
68209 of time.

68210 The *fildev* argument is an open file descriptor associated with a terminal.

68211 If the terminal is not using asynchronous serial data transmission, it is implementation-defined
68212 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any
68213 action.

68214 Attempts to use *tcsendbreak()* from a process which is a member of a background process group
68215 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a
68216 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
68217 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

68218 **RETURN VALUE**

68219 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
68220 indicate the error.

68221 **ERRORS**

68222 The *tcsendbreak()* function shall fail if:

68223 [EBADF] The *fildev* argument is not a valid file descriptor.

68224 [EIO] The process group of the writing process is orphaned, the calling thread is not
68225 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

68226 [ENOTTY] The file associated with *fildev* is not a terminal.

68227 **EXAMPLES**

68228 None.

68229 **APPLICATION USAGE**

68230 None.

68231 **RATIONALE**

68232 None.

68233 **FUTURE DIRECTIONS**

68234 None.

68235 **SEE ALSO**

68236 XBD [Chapter 11](#) (on page 199), [<termios.h>](#)

68237 **CHANGE HISTORY**

68238 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68239 **Issue 6**

68240 The following new requirements on POSIX implementations derive from alignment with the
68241 Single UNIX Specification:

68242 In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now
68243 mandated. This is a FIPS requirement.

68244 The [EIO] error is added.

68245 **Issue 7**

68246 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0649 [79], XSH/TC1-2008/0650 [79],
68247 and XSH/TC1-2008/0651 [79] are applied.

68248 **NAME**

68249 tcsetattr ¶set the parameters associated with the terminal

68250 **SYNOPSIS**

68251 #include <termios.h>

68252 int tcsetattr(int *fildev*, int *optional_actions*,
68253 const struct termios **termios_p*);68254 **DESCRIPTION**68255 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the
68256 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**
68257 structure referenced by *termios_p* as follows:68258 If *optional_actions* is TCSANOW, the change shall occur immediately.68259 If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is
68260 transmitted. This function should be used when changing parameters that affect output.68261 If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is
68262 transmitted, and all input so far received but not read shall be discarded before the change
68263 is made.68264 If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud
68265 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the
68266 line.68267 If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud
68268 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.68269 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested
68270 actions, even if some of the requested actions could not be performed. It shall set all the
68271 attributes that the implementation supports as requested and leave all the attributes not
68272 supported by the implementation unchanged. If no part of the request can be honored, it shall
68273 return -1 and set *errno* to [EINVAL]. If the input and output baud rates differ and are a
68274 combination that is not supported, neither baud rate shall be changed. A subsequent call to
68275 *tcsetattr()* shall return the actual state of the terminal device (reflecting both the changes made
68276 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values
68277 found in the **termios** structure under any circumstances.68278 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios_p*
68279 was not derived from the result of a call to *tcgetattr()* on *fildev*; an application should modify
68280 only fields and flags defined by this volume of POSIX.1-2017 between the call to *tcgetattr()* and
68281 *tcsetattr()*, leaving all other fields and flags unmodified.68282 No actions defined by this volume of POSIX.1-2017, other than a call to *tcsetattr()*, a close of the
68283 last file descriptor in the system associated with this terminal device, or an open of the first file
68284 descriptor in the system associated with this terminal device (using the O_TTY_INIT flag if it is
68285 non-zero and the device is not a pseudo-terminal), shall cause any of the terminal attributes
68286 defined by this volume of POSIX.1-2017 to change.68287 If *tcsetattr()* is called from a process which is a member of a background process group on a *fildev*
68288 associated with its controlling terminal:68289 If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU
68290 signals, the operation completes normally and no signal is sent.

68291 Otherwise, a SIGTTOU signal shall be sent to the process group.

68292 **RETURN VALUE**

68293 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
68294 indicate the error.

68295 **ERRORS**

68296 The *tcsetattr()* function shall fail if:

- | | | |
|-------|----------|---|
| 68297 | [EBADF] | The <i>fildev</i> argument is not a valid file descriptor. |
| 68298 | [EINTR] | A signal interrupted <i>tcsetattr()</i> . |
| 68299 | [EINVAL] | The <i>optional_actions</i> argument is not a supported value, or an attempt was
68300 made to change an attribute represented in the termios structure to an
68301 unsupported value. |
| 68302 | [EIO] | The process group of the writing process is orphaned, the calling thread is not
68303 blocking SIGTTOU, and the process is not ignoring SIGTTOU. |
| 68304 | [ENOTTY] | The file associated with <i>fildev</i> is not a terminal. |

68305 **EXAMPLES**

68306 None.

68307 **APPLICATION USAGE**

68308 If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to
68309 determine what baud rates were actually selected.

68310 In general, there are two reasons for an application to change the parameters associated with a
68311 terminal device:

- 68312 1. The device already has working parameter settings but the application needs a different
68313 behavior, such as non-canonical mode instead of canonical mode. The application sets (or
68314 clears) only a few flags or *c_cc[]* values. Typically, the terminal device in this case is either
68315 the controlling terminal for the process or a pseudo-terminal.
- 68316 2. The device is a modem or similar piece of equipment connected by a serial line, and it
68317 was not open before the application opened it. In this case, the application needs to
68318 initialize all of the parameter settings “from scratch”. However, since the **termios**
68319 structure may include both standard and non-standard parameters, the application
68320 cannot just initialize the whole structure in an arbitrary way (e.g., using *memset()*) as this
68321 may cause some of the non-standard parameters to be set incorrectly, resulting in non-
68322 conforming behavior of the terminal device. Conversely, the application cannot just set
68323 the standard parameters, assuming that the non-standard parameters will already have
68324 suitable values, as the device might previously have been used with non-conforming
68325 parameter settings (and some implementations retain the settings from one use to the
68326 next). The solution is to open the terminal device using the *O_TTY_INIT* flag to initialize
68327 the terminal device to have conforming parameter settings, obtain those settings using
68328 *tcgetattr()*, and then set all of the standard parameters to the desired settings.

68329 **RATIONALE**

68330 The *tcsetattr()* function can be interrupted in the following situations:

68331 It is interrupted while waiting for output to drain.

68332 It is called from a process in a background process group and SIGTTOU is caught.

68333 See also the RATIONALE section in *tcgetattr()*.

68334 FUTURE DIRECTIONS

68335 Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be
68336 supported in a future version of this volume of POSIX.1-2017.

68337 SEE ALSO

68338 *cfgetispeed()*, *tcgetattr()*

68339 XBD Chapter 11 (on page 199), [<termios.h>](#)

68340 CHANGE HISTORY

68341 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68342 Issue 6

68343 The following new requirements on POSIX implementations derive from alignment with the
68344 Single UNIX Specification:

68345 In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now
68346 mandated. This is a FIPS requirement.

68347 The [EIO] error is added.

68348 In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of
68349 a background process group is clarified.

68350 Issue 7

68351 Austin Group Interpretation 1003.1-2001 #144 is applied.

68352 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0652 [79], XSH/TC1-2008/0653 [79],
68353 and XSH/TC1-2008/0654 [79] are applied.

68354 **NAME**

68355 tcsetpgrp — set the foreground process group ID

68356 **SYNOPSIS**

68357 #include <unistd.h>

68358 int tcsetpgrp(int *fildev*, pid_t *pgid_id*);68359 **DESCRIPTION**

68360 If the process has a controlling terminal, *tcsetpgrp()* shall set the foreground process group ID
 68361 associated with the terminal to *pgid_id*. The application shall ensure that the file associated with
 68362 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently
 68363 associated with the session of the calling process. The application shall ensure that the value of
 68364 *pgid_id* matches a process group ID of a process in the same session as the calling process.

68365 Attempts to use *tcsetpgrp()* from a process which is a member of a background process group on
 68366 a *fildev* associated with its controlling terminal shall cause the process group to be sent a
 68367 SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring
 68368 SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

68369 **RETURN VALUE**

68370 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 68371 indicate the error.

68372 **ERRORS**68373 The *tcsetpgrp()* function shall fail if:68374 [EBADF] The *fildev* argument is not a valid file descriptor.68375 [EINVAL] This implementation does not support the value in the *pgid_id* argument.

68376 [EIO] The process group of the writing process is orphaned, the calling thread is not
 68377 blocking SIGTTOU, and the process is not ignoring SIGTTOU.

68378 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
 68379 controlling terminal, or the controlling terminal is no longer associated with
 68380 the session of the calling process.

68381 [EPERM] The value of *pgid_id* is a value supported by the implementation, but does not
 68382 match the process group ID of a process in the same session as the calling
 68383 process.

68384 **EXAMPLES**

68385 None.

68386 **APPLICATION USAGE**

68387 None.

68388 **RATIONALE**

68389 None.

68390 **FUTURE DIRECTIONS**

68391 None.

68392 **SEE ALSO**68393 *tcgetpgrp()*

68394 XBD <sys/types.h>, <unistd.h>

68395 CHANGE HISTORY

68396 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

68397 Issue 6

68398 In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

68399 The following new requirements on POSIX implementations derive from alignment with the
68400 Single UNIX Specification:

68401 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
68402 required for conforming implementations of previous POSIX specifications, it was not
68403 required for UNIX applications.

68404 In the DESCRIPTION and ERRORS sections, text previously conditional on
68405 `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

68406 The normative text is updated to avoid use of the term “must” for application requirements.

68407 The Open Group Corrigendum U047/4 is applied.

68408 Issue 7

68409 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0655 [79] and XSH/TC1-2008/0656
68410 [79] are applied.

68411 **NAME**

68412 tdelete, tfind, tsearch, twalk — manage a binary search tree

68413 **SYNOPSIS**

```
68414 XSI #include <search.h>
68415 void *tdelete(const void *restrict key, void **restrict rootp,
68416             int(*compar)(const void *, const void *));
68417 void *tfind(const void *key, void *const *rootp,
68418            int(*compar)(const void *, const void *));
68419 void *tsearch(const void *key, void **rootp,
68420              int (*compar)(const void *, const void *));
68421 void twalk(const void *root,
68422            void (*action)(const void *, VISIT, int));
```

68423 **DESCRIPTION**

68424 The *tdelete()*, *tfind()*, *tsearch()*, and *twalk()* functions manipulate binary search trees.
 68425 Comparisons are made with a user-supplied routine, the address of which is passed as the
 68426 *compar* argument. This routine is called with two arguments, which are the pointers to the
 68427 elements being compared. The application shall ensure that the user-supplied routine returns an
 68428 integer less than, equal to, or greater than 0, according to whether the first argument is to be
 68429 considered less than, equal to, or greater than the second argument. The comparison function
 68430 need not compare every byte, so arbitrary data may be contained in the elements in addition to
 68431 the values being compared.

68432 The *tsearch()* function shall build and access the tree. The *key* argument is a pointer to an element
 68433 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed
 68434 to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key*
 68435 shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a
 68436 pointer to this node returned. Only pointers are copied, so the application shall ensure that the
 68437 calling routine stores the data. The *rootp* argument points to a variable that points to the root
 68438 node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree;
 68439 in this case, the variable shall be set to point to the node which shall be at the root of the new
 68440 tree.

68441 Like *tsearch()*, *tfind()* shall search for a node in the tree, returning a pointer to it if found.
 68442 However, if it is not found, *tfind()* shall return a null pointer. The arguments for *tfind()* are the
 68443 same as for *tsearch()*.

68444 The *tdelete()* function shall delete a node from a binary search tree. The arguments are the same
 68445 as for *tsearch()*. The variable pointed to by *rootp* shall be changed if the deleted node was the
 68446 root of the tree. If the deleted node was the root of the tree and had no children, the variable
 68447 pointed to by *rootp* shall be set to a null pointer. The *tdelete()* function shall return a pointer to
 68448 the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the
 68449 root node, or a null pointer if the node is not found.

68450 If *tsearch()* adds an element to a tree, or *tdelete()* successfully deletes an element from a tree, the
 68451 concurrent use of that tree in another thread, or use of pointers produced by a previous call to
 68452 *tfind()* or *tsearch()*, produces undefined results.

68453 The *twalk()* function shall traverse a binary search tree. The *root* argument is a pointer to the root
 68454 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below
 68455 that node.) The argument *action* is the name of a routine to be invoked at each node. This routine
 68456 is, in turn, called with three arguments. The first argument shall be the address of the node being
 68457 visited. The structure pointed to by this argument is unspecified and shall not be modified by

68458 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-
 68459 element to access the element stored in the node. The second argument shall be a value from an
 68460 enumeration data type:

```
68461 typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

68462 (defined in <search.h>), depending on whether this is the first, second, or third time that the
 68463 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a
 68464 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

68465 If the calling function alters the pointer to the root, the result is undefined.

68466 If the functions pointed to by *action* or *compar* (for any of these binary search functions) change
 68467 the tree, the results are undefined.

68468 These functions are thread-safe only as long as multiple threads do not access the same tree.

68469 RETURN VALUE

68470 If the node is found, both *tsearch()* and *tfind()* shall return a pointer to it. If not, *tfind()* shall
 68471 return a null pointer, and *tsearch()* shall return a pointer to the inserted item.

68472 A null pointer shall be returned by *tsearch()* if there is not enough space available to create a new
 68473 node.

68474 A null pointer shall be returned by *tdelete()*, *tfind()*, and *tsearch()* if *rootp* is a null pointer on
 68475 entry.

68476 The *tdelete()* function shall return a pointer to the parent of the deleted node, or an unspecified
 68477 non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

68478 The *twalk()* function shall not return a value.

68479 ERRORS

68480 No errors are defined.

68481 EXAMPLES

68482 The following code reads in strings and stores structures containing a pointer to each string and
 68483 a count of its length. It then walks the tree, printing out the stored strings and their lengths in
 68484 alphabetical order.

```
68485 #include <limits.h>
68486 #include <search.h>
68487 #include <stdlib.h>
68488 #include <string.h>
68489 #include <stdio.h>

68490 struct element {          /* Pointers to these are stored in the tree. */
68491     int     count;
68492     char    string[];
68493 };

68494 void *root = NULL;      /* This points to the root. */

68495 int main(void)
68496 {
68497     char    str[_POSIX2_LINE_MAX+1];
68498     int     length = 0;
68499     struct element *elementptr;
68500     void    *node;
68501     void    print_node(const void *, VISIT, int);
```



```

68502     int    node_compare(const void *, const void *),
68503         delete_root(const void *, const void *);

68504     while (fgets(str, sizeof(str), stdin)) {
68505         /* Set element. */
68506         length = strlen(str);
68507         if (str[length-1] == '\n')
68508             str[--length] = '\0';
68509         elementptr = malloc(sizeof(struct element) + length + 1);
68510         strcpy(elementptr->string, str);
68511         elementptr->count = 1;
68512         /* Put element into the tree. */
68513         node = tsearch((void *)elementptr, &root, node_compare);
68514         if (node == NULL) {
68515             fprintf(stderr,
68516                 "tsearch: Not enough space available\n");
68517             exit(EXIT_FAILURE);
68518         }
68519         else if (*(struct element **)node != elementptr) {
68520             /* A node containing the element already exists */
68521             (*(struct element **)node)->count++;
68522             free(elementptr);
68523         }
68524     }
68525     twalk(root, print_node);

68526     /* Delete all nodes in the tree */
68527     while (root != NULL) {
68528         elementptr = *(struct element **)root;
68529         printf("deleting node: string = %s, count = %d\n",
68530             elementptr->string,
68531             elementptr->count);
68532         tdelete((void *)elementptr, &root, delete_root);
68533         free(elementptr);
68534     }

68535     return 0;
68536 }

68537 /*
68538  * This routine compares two nodes, based on an
68539  * alphabetical ordering of the string field.
68540  */
68541 int
68542 node_compare(const void *node1, const void *node2)
68543 {
68544     return strcmp(((const struct element *) node1)->string,
68545                 ((const struct element *) node2)->string);
68546 }

68547 /*
68548  * This comparison routine can be used with tdelete()
68549  * when explicitly deleting a root node, as no comparison
68550  * is necessary.

```

```

68551     */
68552     int
68553     delete_root(const void *node1, const void *node2)
68554     {
68555         return 0;
68556     }
68557     /*
68558     * This routine prints out a node, the second time
68559     * twalk encounters it or if it is a leaf.
68560     */
68561     void
68562     print_node(const void *ptr, VISIT order, int level)
68563     {
68564         const struct element *p = *(const struct element **) ptr;
68565         if (order == postorder || order == leaf) {
68566             (void) printf("string = %s, count = %d\n",
68567                 p->string, p->count);
68568         }
68569     }

```

68570 APPLICATION USAGE

68571 The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()*
68572 and *tsearch()*.

68573 There are two nomenclatures used to refer to the order in which tree nodes are visited. The
68574 *twalk()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node
68575 before any of its children, after its left child and before its right, and after both its children. The
68576 alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which
68577 could result in some confusion over the meaning of **postorder**.

68578 Since the return value of *tdelete()* is an unspecified non-null pointer in the case that the root of
68579 the tree has been deleted, applications should only use the return value of *tdelete()* as indication
68580 of success or failure and should not assume it can be dereferenced. Some implementations in this
68581 case will return a pointer to the new root of the tree (or to an empty tree if the deleted root node
68582 was the only node in the tree); other implementations return arbitrary non-null pointers.

68583 RATIONALE

68584 None.

68585 FUTURE DIRECTIONS

68586 None.

68587 SEE ALSO

68588 [hcreate\(\)](#), [lsearch\(\)](#)

68589 XBD [<search.h>](#)

68590 CHANGE HISTORY

68591 First released in Issue 1. Derived from Issue 1 of the SVID.

68592 Issue 5

68593 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
68594 previous issues.

68595 Issue 6

68596 The normative text is updated to avoid use of the term “must” for application requirements.

68597 The **restrict** keyword is added to the *tdelete()* prototype for alignment with the
68598 ISO/IEC 9899:1999 standard.

68599 Issue 7

68600 Austin Group Interpretation 1003.1-2001 #149 is applied, clarifying concurrent use of the tree in
68601 another thread.

68602 Austin Group Interpretation 1003.1-2001 #151 is applied, clarifying behavior for *tdelete()* when
68603 the deleted node is the root node.

68604 Austin Group Interpretation 1003.1-2001 #153 is applied.

68605 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0366 [551] is applied.

68606 **NAME**

68607 tellmdir — current location of a named directory stream

68608 **SYNOPSIS**

```
68609 XSI #include <dirent.h>  
68610 long tellmdir(DIR *dirp);
```

68611 **DESCRIPTION**

68612 The *tellmdir()* function shall obtain the current location associated with the directory stream
68613 specified by *dirp*.

68614 If the most recent operation on the directory stream was a *seekdir()*, the directory position
68615 returned from the *tellmdir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.

68616 **RETURN VALUE**

68617 Upon successful completion, *tellmdir()* shall return the current location of the specified directory
68618 stream.

68619 **ERRORS**

68620 No errors are defined.

68621 **EXAMPLES**

68622 None.

68623 **APPLICATION USAGE**

68624 None.

68625 **RATIONALE**

68626 None.

68627 **FUTURE DIRECTIONS**

68628 None.

68629 **SEE ALSO**

68630 *fdopendir()*, *readdir()*, *seekdir()*

68631 XBD <dirent.h>

68632 **CHANGE HISTORY**

68633 First released in Issue 2.

68634 **NAME**

68635 tempnam — create a name for a temporary file

68636 **SYNOPSIS**

```
68637 OB XSI #include <stdio.h>
68638 char *tempnam(const char *dir, const char *pfx);
```

68639 **DESCRIPTION**68640 The *tempnam()* function shall generate a pathname that may be used for a temporary file.

68641 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument
 68642 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or
 68643 points to a string which is not a name for an appropriate directory, the path prefix defined as
 68644 P_tmpdir in the <stdio.h> header shall be used. If that directory is not accessible, an
 68645 implementation-defined directory may be used.

68646 Many applications prefer their temporary files to have certain initial letter sequences in their
 68647 names. The *pfx* argument should be used for this. This argument may be a null pointer or point
 68648 to a string of up to five bytes to be used as the beginning of the filename.

68649 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if
 68650 called more than {TMP_MAX} times in a single process, the behavior is implementation-defined.

68651 **RETURN VALUE**

68652 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated
 68653 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a
 68654 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the
 68655 error.

68656 **ERRORS**68657 The *tempnam()* function shall fail if:

68658 [ENOMEM] Insufficient storage space is available.

68659 **EXAMPLES**68660 **Generating a Pathname**

68661 The following example generates a pathname for a temporary file in directory */tmp*, with the
 68662 prefix *file*. After the pathname has been created, the call to *free()* deallocates the space used to
 68663 store the pathname.

```
68664 #include <stdio.h>
68665 #include <stdlib.h>
68666 ...
68667 const char *directory = "/tmp";
68668 const char *fileprefix = "file";
68669 char *file;

68670 file = tempnam(directory, fileprefix);
68671 free(file);
```

68672 **APPLICATION USAGE**

68673 This function only creates pathnames. It is the application's responsibility to create and remove
 68674 the files. Between the time a pathname is created and the file is opened, it is possible for some
 68675 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

68676 Applications should use the *tmpfile()*, *mkdtemp()*, or *mkstemp()* functions instead of the

68677 obsolescent *tempnam()* function.

68678 **RATIONALE**

68679 None.

68680 **FUTURE DIRECTIONS**

68681 The *tempnam()* function may be removed in a future version.

68682 **SEE ALSO**

68683 *fopen()*, *free()*, *mkdtemp()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink()*

68684 XBD <**stdio.h**>

68685 **CHANGE HISTORY**

68686 First released in Issue 1. Derived from Issue 1 of the SVID.

68687 **Issue 5**

68688 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
68689 previous issues.

68690 **Issue 7**

68691 The *tempnam()* function is marked obsolescent.

68692 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0657 [291], XSH/TC1-2008/0658 [137],
68693 and XSH/TC1-2008/0659 [137] are applied.

68694 **NAME**

68695 tfind — search binary search tree

68696 **SYNOPSIS**

```
68697 XSI       #include <search.h>
68698       void *tfind(const void *key, void *const *rootp,
68699                   int (*compar)(const void *, const void *));
```

68700 **DESCRIPTION**68701 Refer to *tdelete()*.

68702 **NAME**

68703 tgamma, tgammaf, tgammal †compute gamma() function

68704 **SYNOPSIS**

```
68705 #include <math.h>
68706 double tgamma(double x);
68707 float tgammaf(float x);
68708 long double tgammal(long double x);
```

68709 **DESCRIPTION**

68710 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68711 conflict between the requirements described here and the ISO C standard is unintentional. This
 68712 volume of POSIX.1-2017 defers to the ISO C standard.

68713 These functions shall compute $\Gamma(x)$ where $\Gamma(x)$ is defined as $\int_0^{\infty} e^{-t} t^{x-1} dt$.

68714 An application wishing to check for error situations should set *errno* to zero and call
 68715 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 68716 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 68717 zero, an error has occurred.

68718 **RETURN VALUE**68719 Upon successful completion, these functions shall return the gamma of *x*.

68720 CX If *x* is a negative integer, a domain error may occur and either a NaN (if supported) or an
 68721 MX implementation-defined value shall be returned. On systems that support the IEC 60559
 68722 Floating-Point option, a domain error shall occur and a NaN shall be returned.

68723 If *x* is ± 0 , *tgamma()*, *tgammaf()*, and *tgammal()* shall return \pm HUGE_VAL, \pm HUGE_VALF, and
 68724 MX \pm HUGE_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, a
 68725 CX pole error shall occur; otherwise, a pole error may occur.

68726 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,
 68727 and *tgammal()* shall return \pm HUGE_VAL, \pm HUGE_VALF, or \pm HUGE_VALL, respectively, with
 68728 the same sign as the correct value of the function.

68729 MX If the correct value would cause underflow, and is not representable, a range error may occur,
 68730 MX and *tgamma()*, *tgammaf()*, and *tgammal()* shall return 0.0, or (if IEC 60559 Floating-Point is not
 68731 supported) an implementation-defined value no greater in magnitude than DBL_MIN,
 68732 FLT_MIN, and LDBL_MIN, respectively.

68733 MX If the correct value would cause underflow, and is representable, a range error may occur and
 68734 the correct value shall be returned.

68735 If *x* is subnormal and $1/x$ is representable, $1/x$ should be returned.68736 MX If *x* is NaN, a NaN shall be returned.68737 If *x* is +Inf, *x* shall be returned.68738 If *x* is -Inf, a domain error shall occur, and a NaN shall be returned.68739 **ERRORS**

68740 These functions shall fail if:

68741 MX **Domain Error** The value of *x* is a negative integer, or *x* is -Inf.

68742 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 68743 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*

68744 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
68745 shall be raised.

68746 MX Pole Error The value of x is zero.

68747 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
68748 then *errno* shall be set to [ERANGE]. If the integer expression
68749 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
68750 floating-point exception shall be raised.

68751 Range Error The value overflows.

68752 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
68753 then *errno* shall be set to [ERANGE]. If the integer expression
68754 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
68755 floating-point exception shall be raised.

68756 These functions may fail if:

68757 Domain Error The value of x is a negative integer.

68758 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
68759 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
68760 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
68761 shall be raised.

68762 Pole Error The value of x is zero.

68763 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
68764 then *errno* shall be set to [ERANGE]. If the integer expression
68765 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
68766 floating-point exception shall be raised.

68767 Range Error The result underflows.

68768 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
68769 then *errno* shall be set to [ERANGE]. If the integer expression
68770 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
68771 floating-point exception shall be raised.

68772 EXAMPLES

68773 None.

68774 APPLICATION USAGE

68775 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
68776 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

68777 RATIONALE

68778 This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and
68779 *lgamma()* functions.

68780 FUTURE DIRECTIONS

68781 It is possible that the error response for a negative integer argument may be changed to a pole
68782 error and a return value of $\pm\text{Inf}$.

68783 SEE ALSO

68784 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [lgamma\(\)](#)

68785 XBD Section 4.20 (on page 117), [<math.h>](#)

CHANGE HISTORY

- 68786
68787 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.
- 68788
68789 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third paragraph in the RETURN VALUE section.
- 68790 **Issue 7**
- 68791 ISO/IEC 9899: 1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.
- 68792
68793 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0660 [68], XSH/TC1-2008/0661 [320], and XSH/TC1-2008/0662 [68] are applied.
- 68794
68795 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0367 [604] and XSH/TC2-2008/0368 [630] are applied.

68796 **NAME**

68797 time ‡get time

68798 **SYNOPSIS**

68799 #include <time.h>

68800 time_t time(time_t *tloc);

68801 **DESCRIPTION**

68802 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68803 conflict between the requirements described here and the ISO C standard is unintentional. This
 68804 volume of POSIX.1-2017 defers to the ISO C standard.

68805 CX The *time()* function shall return the value of time in seconds since the Epoch.

68806 The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer,
 68807 no value is stored.

68808 **RETURN VALUE**

68809 Upon successful completion, *time()* shall return the value of time. Otherwise, (**time_t**)-1 shall be
 68810 returned.

68811 **ERRORS**68812 The *time()* function may fail if:

68813 CX **[EOVERFLOW]** The number of seconds since the Epoch will not fit in an object of type **time_t**.

68814 **EXAMPLES**68815 **Getting the Current Time**

68816 The following example uses the *time()* function to calculate the time elapsed, in seconds, since
 68817 the Epoch, *localtime()* to convert that value to a broken-down time, and *asctime()* to convert the
 68818 broken-down time values into a printable string.

```
68819 #include <stdio.h>
68820 #include <time.h>
68821 int main(void)
68822 {
68823     time_t result;
68824     result = time(NULL);
68825     printf("%s%ju secs since the Epoch\n",
68826           asctime(localtime(&result)),
68827           (uintmax_t)result);
68828     return(0);
68829 }
```

68830 This example writes the current time to *stdout* in a form like this:

```
68831 Wed Jun 26 10:32:15 1996
68832 835810335 secs since the Epoch
```

68833 **Timing an Event**

68834 The following example gets the current time, prints it out in the user's format, and prints the
68835 number of minutes to an event being timed.

```
68836 #include <time.h>
68837 #include <stdio.h>
68838 ...
68839 time_t now;
68840 int minutes_to_event;
68841 ...
68842 time(&now);
68843 minutes_to_event = ...;
68844 printf("The time is ");
68845 puts(asctime(localtime(&now)));
68846 printf("There are %d minutes to the event.\n",
68847        minutes_to_event);
68848 ...
```

68849 **APPLICATION USAGE**

68850 None.

68851 **RATIONALE**

68852 The *time()* function returns a value in seconds while *clock_gettime()* and *gettimeofday()* return a
68853 **struct timespec** (seconds and nanoseconds) and **struct timeval** (seconds and microseconds),
68854 respectively, and are therefore capable of returning more precise times. The *times()* function is
68855 also capable of more precision than *time()* as it returns a value in clock ticks, although it returns
68856 the elapsed time since an arbitrary point such as system boot time, not since the epoch.

68857 Implementations in which **time_t** is a 32-bit signed integer (many historical implementations)
68858 fail in the year 2038. POSIX.1-2017 does not address this problem. However, the use of the **time_t**
68859 type is mandated in order to ease the eventual fix.

68860 On some systems the *time()* function is implemented using a system call that does not return an
68861 error condition in addition to the return value. On these systems it is impossible to differentiate
68862 between valid and invalid return values and hence overflow conditions cannot be reliably
68863 detected.

68864 The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C
68865 standard.

68866 Many historical implementations (including Version 7) and the 1984 /usr/group standard use
68867 **long** instead of **time_t**. This volume of POSIX.1-2017 uses the latter type in order to agree with
68868 the ISO C standard.

68869 **FUTURE DIRECTIONS**

68870 In a future version of this volume of POSIX.1-2017, **time_t** is likely to be required to be capable
68871 of representing times far in the future. Whether this will be mandated as a 64-bit type or a
68872 requirement that a specific date in the future be representable (for example, 10000 AD) is not yet
68873 determined. Systems purchased after the approval of this volume of POSIX.1-2017 should be
68874 evaluated to determine whether their lifetime will extend past 2038.

68875 **SEE ALSO**

68876 *asctime()*, *clock()*, *clock_getres()*, *ctime()*, *difftime()*, *futimens()*, *gettimeofday()*, *gmtime()*,
68877 *localtime()*, *mktime()*, *strftime()*, *strptime()*, *times()*, *utime()*

68878 XBD **<time.h>**

68879 CHANGE HISTORY

68880 First released in Issue 1. Derived from Issue 1 of the SVID.

68881 Issue 6

68882 Extensions beyond the ISO C standard are marked.

68883 The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.

68884 Issue 7

68885 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0663 [106], XSH/TC1-2008/0664 [350],
68886 XSH/TC1-2008/0665 [106], XSH/TC1-2008/0666 [350], and XSH/TC1-2008/0667 [350] are
68887 applied.

68888 **NAME**

68889 timer_create — create a per-process timer

68890 **SYNOPSIS**

```
68891 CX #include <signal.h>
68892 #include <time.h>
68893 int timer_create(clockid_t clockid, struct sigevent *restrict evp,
68894 timer_t *restrict timerid);
```

68895 **DESCRIPTION**

68896 The *timer_create()* function shall create a per-process timer using the specified clock, *clock_id*, as
 68897 the timing base. The *timer_create()* function shall return, in the location referenced by *timerid*, a
 68898 timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be
 68899 unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is
 68900 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return
 68901 from *timer_create()*.

68902 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the
 68903 application, defines the asynchronous notification to occur as specified in [Section 2.4.1](#) (on page
 68904 488) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument
 68905 pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL,
 68906 the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of
 68907 the timer ID.

68908 Each implementation shall define a set of clocks that can be used as timing bases for per-process
 68909 **MON** timers. All implementations shall support a *clock_id* of CLOCK_REALTIME. If the Monotonic
 68910 Clock option is supported, implementations shall support a *clock_id* of CLOCK_MONOTONIC.

68911 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed
 68912 and deleted by an *exec*.

68913 **CPT** If _POSIX_CPUTIME is defined, implementations shall support *clock_id* values representing the
 68914 CPU-time clock of the calling process.

68915 **TCT** If _POSIX_THREAD_CPUTIME is defined, implementations shall support *clock_id* values
 68916 representing the CPU-time clock of the calling thread.

68917 **CPT|TCT** It is implementation-defined whether a *timer_create()* function will succeed if the value defined
 68918 by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process
 68919 or thread invoking the function.

68920 **TSA** If *evp* *sigev_notify* is SIGEV_THREAD and *sev* *sigev_notify_attributes* is not NULL, if the
 68921 attribute pointed to by *sev* *sigev_notify_attributes* has a thread stack address specified by a call
 68922 to *pthread_attr_setstack()*, the results are unspecified if the signal is generated more than once.

68923 **RETURN VALUE**

68924 If the call succeeds, *timer_create()* shall return zero and update the location referenced by *timerid*
 68925 to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function
 68926 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if
 68927 an error occurs.

68928 **ERRORS**68929 The *timer_create()* function shall fail if:

68930 [EAGAIN] The system lacks sufficient signal queuing resources to honor the request.

68931 [EAGAIN] The calling process has already created all of the timers it is allowed by this
68932 implementation.

68933 [EINVAL] The specified clock ID is not defined.

68934 CPT|TCT [ENOTSUP] The implementation does not support the creation of a timer attached to the
68935 CPU-time clock that is specified by *clock_id* and associated with a process or
68936 thread different from the process or thread invoking *timer_create()*.

68937 EXAMPLES

68938 None.

68939 APPLICATION USAGE

68940 If a timer is created which has *evp -sigev_sigev_notify* set to SIGEV_THREAD and the attribute
68941 pointed to by *evp -sigev_notify_attributes* has a thread stack address specified by a call to
68942 *pthread_attr_setstack()*, the memory dedicated as a thread stack cannot be recovered. The reason
68943 for this is that the threads created in response to a timer expiration are created detached, or in an
68944 unspecified way if the thread attribute's *detachstate* is PTHREAD_CREATE_JOINABLE. In
68945 neither case is it valid to call *pthread_join()*, which makes it impossible to determine the lifetime
68946 of the created thread which thus means the stack memory cannot be reused.

68947 RATIONALE**68948 Periodic Timer Overrun and Resource Allocation**

68949 The specified timer facilities may deliver realtime signals (that is, queued signals) on
68950 implementations that support this option. Since realtime applications cannot afford to lose
68951 notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it
68952 must be possible to ensure that sufficient resources exist to deliver the signal when the event
68953 occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a
68954 request and a subsequent signal generation. If the request cannot allocate the signal delivery
68955 resources, it can fail the call with an [EAGAIN] error.

68956 Periodic timers are a special case. A single request can generate an unspecified number of
68957 signals. This is not a problem if the requesting process can service the signals as fast as they are
68958 generated, thus making the signal delivery resources available for delivery of subsequent
68959 periodic timer expiration signals. But, in general, this cannot be assured — processing of periodic
68960 timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the
68961 currently pending signal has been delivered.

68962 Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a
68963 pending signal are generated, it is implementation-defined whether a signal is delivered for each
68964 occurrence. This is not adequate for some realtime applications. So a mechanism is required to
68965 allow applications to detect how many timer expirations were delayed without requiring an
68966 indefinite amount of system resources to store the delayed expirations.

68967 The specified facilities provide for an overrun count. The overrun count is defined as the
68968 number of extra timer expirations that occurred between the time a timer expiration signal is
68969 generated and the time the signal is delivered. The signal-catching function, if it is concerned
68970 with overruns, can retrieve this count on entry. With this method, a periodic timer only needs
68971 one “signal queuing resource” that can be allocated at the time of the *timer_create()* function call.

68972 A function is defined to retrieve the overrun count so that an application need not allocate static
68973 storage to contain the count, and an implementation need not update this storage
68974 asynchronously on timer expirations. But, for some high-frequency periodic applications, the
68975 overhead of an additional system call on each timer expiration may be prohibitive. The
68976 functions, as defined, permit an implementation to maintain the overrun count in user space,

68977 associated with the *timerid*. The *timer_getoverrun()* function can then be implemented as a macro
 68978 that uses the *timerid* argument (which may just be a pointer to a user space structure containing
 68979 the counter) to locate the overrun count with no system call overhead. Other implementations,
 68980 less concerned with this class of applications, can avoid the asynchronous update of user space
 68981 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

68982 **Timer Expiration Signal Parameters**

68983 The Realtime Signals Extension option supports an application-specific datum that is delivered
 68984 to the extended signal handler. This value is explicitly specified by the application, along with
 68985 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined
 68986 value can be either an integer constant or a pointer. This explicit specification of the value, as
 68987 opposed to always sending the timer ID, was selected based on existing practice.

68988 It is common practice for realtime applications (on non-POSIX systems or realtime extended
 68989 POSIX systems) to use the parameters of event handlers as the case label of a switch statement or
 68990 as a pointer to an application-defined data structure. Since *timer_ids* are dynamically allocated
 68991 by the *timer_create()* function, they can be used for neither of these functions without additional
 68992 application overhead in the signal handler; for example, to search an array of saved timer IDs to
 68993 associate the ID with a constant or application data structure.

68994 **FUTURE DIRECTIONS**

68995 None.

68996 **SEE ALSO**

68997 [clock_getres\(\)](#), [timer_delete\(\)](#), [timer_getoverrun\(\)](#)

68998 XBD [<signal.h>](#), [<time.h>](#)

68999 **CHANGE HISTORY**

69000 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

69001 **Issue 6**

69002 The *timer_create()* function is marked as part of the Timers option.

69003 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 69004 implementation does not support the Timers option.

69005 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

69006 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the
 69007 requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

69008 The **restrict** keyword is added to the *timer_create()* prototype for alignment with the
 69009 ISO/IEC 9899:1999 standard.

69010 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/138 is applied, updating the
 69011 DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created
 69012 with the notification method set to SIGEV_THREAD.

69013 **Issue 7**

69014 The *timer_create()* function is moved from the Timers option to the Base.

69015 **NAME**

69016 timer_delete — delete a per-process timer

69017 **SYNOPSIS**

```
69018 CX #include <time.h>
69019 int timer_delete(timer_t timerid);
```

69020 **DESCRIPTION**

69021 The *timer_delete()* function deletes the specified timer, *timerid*, previously created by the
69022 *timer_create()* function. If the timer is armed when *timer_delete()* is called, the behavior shall be
69023 as if the timer is automatically disarmed before removal. The disposition of pending signals for
69024 the deleted timer is unspecified.

69025 The behavior is undefined if the value specified by the *timerid* argument to *timer_delete()* does
69026 not correspond to a timer ID returned by *timer_create()* but not yet deleted by *timer_delete()*.

69027 **RETURN VALUE**

69028 If successful, the *timer_delete()* function shall return a value of zero. Otherwise, the function shall
69029 return a value of -1 and set *errno* to indicate the error.

69030 **ERRORS**

69031 No errors are defined.

69032 **EXAMPLES**

69033 None.

69034 **APPLICATION USAGE**

69035 None.

69036 **RATIONALE**

69037 If an implementation detects that the value specified by the *timerid* argument to *timer_delete()*
69038 does not correspond to a timer ID returned by *timer_create()* but not yet deleted by
69039 *timer_delete()*, it is recommended that the function should fail and report an [EINVAL] error.

69040 **FUTURE DIRECTIONS**

69041 None.

69042 **SEE ALSO**69043 [*timer_create\(\)*](#)69044 XBD [<time.h>](#)69045 **CHANGE HISTORY**

69046 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

69047 **Issue 6**69048 The *timer_delete()* function is marked as part of the Timers option.

69049 The [ENOSYS] error condition has been removed as stubs need not be provided if an
69050 implementation does not support the Timers option.

69051 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/139 is applied, updating the ERRORS
69052 section so that the [EINVAL] error becomes optional.

69053 **Issue 7**69054 The *timer_delete()* function is moved from the Timers option to the Base.

69055 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0369 [659] is applied.

69056 **NAME**

69057 timer_getoverrun, timer_gettime, timer_settime — per-process timers

69058 **SYNOPSIS**

```

69059 CX #include <time.h>
69060
69061 int timer_getoverrun(timer_t timerid);
69062 int timer_gettime(timer_t timerid, struct itimerspec *value);
69063 int timer_settime(timer_t timerid, int flags,
69064                  const struct itimerspec *restrict value,
69065                  struct itimerspec *restrict ovalue);

```

69065 **DESCRIPTION**

69066 The *timer_gettime()* function shall store the amount of time until the specified timer, *timerid*,
 69067 expires and the reload value of the timer into the space pointed to by the *value* argument. The
 69068 *it_value* member of this structure shall contain the amount of time before the timer expires, or
 69069 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if
 69070 the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload
 69071 value last set by *timer_settime()*.

69072 The *timer_settime()* function shall set the time until the next expiration of the timer specified by
 69073 *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member
 69074 of *value* is non-zero. If the specified timer was already armed when *timer_settime()* is called, this
 69075 call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value*
 69076 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending
 69077 expiration notifications is unspecified.

69078 If the flag `TIMER_ABSTIME` is not set in the argument *flags*, *timer_settime()* shall behave as if the
 69079 time until next expiration is set to be equal to the interval specified by the *it_value* member of
 69080 *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the
 69081 flag `TIMER_ABSTIME` is set in the argument *flags*, *timer_settime()* shall behave as if the time
 69082 until next expiration is set to be equal to the difference between the absolute time specified by
 69083 the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is,
 69084 the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*.
 69085 If the specified time has already passed, the function shall succeed and the expiration
 69086 notification shall be made.

69087 The reload value of the timer shall be set to the value specified by the *it_interval* member of
 69088 *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is
 69089 specified.

69090 Time values that are between two consecutive non-negative integer multiples of the resolution of
 69091 the specified timer shall be rounded up to the larger multiple of the resolution. Quantization
 69092 error shall not cause the timer to expire earlier than the rounded time value.

69093 If the argument *ovalue* is not `NULL`, the *timer_settime()* function shall store, in the location
 69094 referenced by *ovalue*, a value representing the previous amount of time before the timer would
 69095 have expired, or zero if the timer was disarmed, together with the previous timer reload value.
 69096 Timers shall not expire before their scheduled time.

69097 Only a single signal shall be queued to the process for a given timer at any point in time. When a
 69098 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun
 69099 shall occur. When a timer expiration signal is delivered to or accepted by a process, the
 69100 *timer_getoverrun()* function shall return the timer expiration overrun count for the specified
 69101 timer. The overrun count returned contains the number of extra timer expirations that occurred
 69102 between the time the signal was generated (queued) and when it was delivered or accepted, up

69103 to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the
 69104 number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the
 69105 overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun()*
 69106 shall apply to the most recent expiration signal delivery or acceptance for the timer. If no
 69107 expiration signal has been delivered for the timer, the return value of *timer_getoverrun()* is
 69108 unspecified.

69109 The behavior is undefined if the value specified by the *timerid* argument to *timer_getoverrun()*,
 69110 *timer_gettime()*, or *timer_settime()* does not correspond to a timer ID returned by *timer_create()*
 69111 but not yet deleted by *timer_delete()*.

69112 RETURN VALUE

69113 If the *timer_getoverrun()* function succeeds, it shall return the timer expiration overrun count as
 69114 explained above.

69115 If the *timer_gettime()* or *timer_settime()* functions succeed, a value of 0 shall be returned.

69116 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to
 69117 indicate the error.

69118 ERRORS

69119 The *timer_settime()* function shall fail if:

69120 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than
 69121 or equal to 1 000 million, and the *it_value* member of that structure did not
 69122 specify zero seconds and nanoseconds.

69123 The *timer_settime()* function may fail if:

69124 [EINVAL] The *it_interval* member of *value* is not zero and the timer was created with
 69125 notification by creation of a new thread (*sigev_sigev_notify* was
 69126 SIGEV_THREAD) and a fixed stack address has been set in the thread
 69127 attribute pointed to by *sigev_notify_attributes*.

69128 EXAMPLES

69129 None.

69130 APPLICATION USAGE

69131 Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a
 69132 new thread. Since it cannot be assumed that the thread created for one expiration is finished
 69133 before the next expiration of the timer, it could happen that two threads use the same memory as
 69134 a stack at the same time. This is invalid and produces undefined results.

69135 RATIONALE

69136 Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The
 69137 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either
 69138 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for
 69139 these common rates. The granularity of practical clocks implies that if one reads a given clock
 69140 twice in rapid succession, one may get the same time value twice; and that timers must wait for
 69141 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too
 69142 soon. Note also that the granularity of the clock may be significantly coarser than the resolution
 69143 of the data format used to set and get time and interval values. Also note that some
 69144 implementations may choose to adjust time and/or interval values to exactly match the ticks of
 69145 the underlying clock.

69146 This volume of POSIX.1-2017 defines functions that allow an application to determine the
 69147 implementation-supported resolution for the clocks and requires an implementation to
 69148 document the resolution supported for timers and *nanosleep()* if they differ from the supported

- 69149 clock resolution. This is more of a procurement issue than a runtime application issue.
- 69150 If an implementation detects that the value specified by the *timerid* argument to
69151 *timer_getoverrun()*, *timer_gettime()*, or *timer_settime()* does not correspond to a timer ID returned
69152 by *timer_create()* but not yet deleted by *timer_delete()*, it is recommended that the function
69153 should fail and report an [EINVAL] error.
- 69154 **FUTURE DIRECTIONS**
- 69155 None.
- 69156 **SEE ALSO**
- 69157 *clock_getres()*, *timer_create()*
- 69158 XBD <[time.h](#)>
- 69159 **CHANGE HISTORY**
- 69160 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
- 69161 **Issue 6**
- 69162 The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are marked as part of the
69163 Timers option.
- 69164 The [ENOSYS] error condition has been removed as stubs need not be provided if an
69165 implementation does not support the Timers option.
- 69166 The [EINVAL] error condition is updated to include the following: “and the *it_value* member of
69167 that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC
69168 Interpretation 1003.1 #89.
- 69169 The DESCRIPTION for *timer_getoverrun()* is updated to clarify that “If no expiration signal has
69170 been delivered for the timer, or if the Realtime Signals Extension is not supported, the return
69171 value of *timer_getoverrun()* is unspecified”.
- 69172 The **restrict** keyword is added to the *timer_settime()* prototype for alignment with the
69173 ISO/IEC 9899:1999 standard.
- 69174 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/140 is applied, updating the ERRORS
69175 section so that the mandatory [EINVAL] error (“The *timerid* argument does not correspond to an
69176 ID returned by *timer_create()* but not yet deleted by *timer_delete()*”) becomes optional.
- 69177 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/141 is applied, updating the ERRORS
69178 section to include an optional [EINVAL] error for the case when a timer is created with the
69179 notification method set to SIGEV_THREAD. APPLICATION USAGE text is also added.
- 69180 **Issue 7**
- 69181 The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are moved from the Timers
69182 option to the Base.
- 69183 Functionality relating to the Realtime Signals Extension option is moved to the Base.
- 69184 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0370 [659] is applied.

69185 **NAME**

69186 times — get process and waited-for child process times

69187 **SYNOPSIS**

```
69188 #include <sys/times.h>
69189 clock_t times(struct tms *buffer);
```

69190 **DESCRIPTION**

69191 The *times()* function shall fill the **tms** structure pointed to by *buffer* with time-accounting
 69192 information. The **tms** structure is defined in **<sys/times.h>**.

69193 All times are measured in terms of the number of clock ticks used.

69194 The times of a terminated child process shall be included in the *tms_cutime* and *tms_cstime*
 69195 elements of the parent when *wait()*, *waitid()*, or *waitpid()* returns the process ID of this
 69196 terminated child. If a child process has not waited for its children, their times shall not be
 69197 included in its times.

69198 The *tms_utime* structure member is the CPU time charged for the execution of user
 69199 instructions of the calling process.

69200 The *tms_stime* structure member is the CPU time charged for execution by the system on
 69201 behalf of the calling process.

69202 The *tms_cutime* structure member is the sum of the *tms_utime* and *tms_cutime* times of the
 69203 child processes.

69204 The *tms_cstime* structure member is the sum of the *tms_stime* and *tms_cstime* times of the
 69205 child processes.

69206 **RETURN VALUE**

69207 Upon successful completion, *times()* shall return the elapsed real time, in clock ticks, since an
 69208 arbitrary point in the past (for example, system start-up time). This point does not change from
 69209 one invocation of *times()* within the process to another. The return value may overflow the
 69210 possible range of type **clock_t**. If *times()* fails, **(clock_t)-1** shall be returned and *errno* set to
 69211 indicate the error.

69212 **ERRORS**

69213 The *times()* function shall fail if:

69214 [EOVERFLOW] The return value would overflow the range of **clock_t**.

69215 **EXAMPLES**69216 **Timing a Database Lookup**

69217 The following example defines two functions, *start_clock()* and *end_clock()*, that are used to time
 69218 a lookup. It also defines variables of type **clock_t** and **tms** to measure the duration of
 69219 transactions. The *start_clock()* function saves the beginning times given by the *times()* function.
 69220 The *end_clock()* function gets the ending times and prints the difference between the two times.

```
69221 #include <sys/times.h>
69222 #include <stdio.h>
69223 ...
69224 void start_clock(void);
69225 void end_clock(char *msg);
69226 ...
69227 static clock_t st_time;
69228 static clock_t en_time;
```

```

69229     static struct tms st_cpu;
69230     static struct tms en_cpu;
69231     ...
69232     void
69233     start_clock()
69234     {
69235         st_time = times(&st_cpu);
69236     }

69237     /* This example assumes that the result of each subtraction
69238        is within the range of values that can be represented in
69239        an integer type. */
69240     void
69241     end_clock(char *msg)
69242     {
69243         en_time = times(&en_cpu);

69244         fputs(msg, stdout);
69245         printf("Real Time: %jd, User Time %jd, System Time %jd\n",
69246             (intmax_t)(en_time - st_time),
69247             (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
69248             (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
69249     }

```

69250 APPLICATION USAGE

69251 Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per
69252 second as it may vary from system to system.

69253 RATIONALE

69254 The accuracy of the times reported is intentionally left unspecified to allow implementations
69255 flexibility in design, from uniprocessor to multi-processor networks.

69256 The inclusion of times of child processes is recursive, so that a parent process may collect the
69257 total times of all of its descendants. But the times of a child are only added to those of its parent
69258 when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process
69259 can always see the total times of all its descendants; see also the discussion of the term
69260 “realtime” in `alarm()`.

69261 If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a
69262 year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual
69263 systems that run continuously for longer than that. This volume of POSIX.1-2017 permits an
69264 implementation to make the reference point for the returned value be the start-up time of the
69265 process, rather than system start-up time.

69266 The term “charge” in this context has nothing to do with billing for services. The operating
69267 system accounts for time used in this way. That information must be correct, regardless of how
69268 that information is used.

69269 FUTURE DIRECTIONS

69270 None.

69271 SEE ALSO

69272 `alarm()`, `exec`, `fork()`, `sysconf()`, `time()`, `wait()`, `waitid()`

69273 XBD `<sys/times.h>`

69274 **CHANGE HISTORY**

69275 First released in Issue 1. Derived from Issue 1 of the SVID.

69276 **Issue 7**

69277 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0371 [644] is applied.

69278 **NAME**

69279 timezone — difference from UTC and local standard time

69280 **SYNOPSIS**

```
69281 XSI        #include <time.h>
69282            extern long timezone;
```

69283 **DESCRIPTION**69284 Refer to [tzset\(\)](#).

69285 **NAME**

69286 tmpfile — create a temporary file

69287 **SYNOPSIS**

69288 #include <stdio.h>

69289 FILE *tmpfile(void);

69290 **DESCRIPTION**

69291 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69292 conflict between the requirements described here and the ISO C standard is unintentional. This
 69293 volume of POSIX.1-2017 defers to the ISO C standard.

69294 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file
 69295 shall be automatically deleted when all references to the file are closed. The file shall be opened
 69296 as in *fopen()* for update (*wb+*), except that implementations may restrict the permissions, either
 69297 by clearing the file mode bits or setting them to the value *S_IRUSR | S_IWUSR*.

69298 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is
 69299 killed while it is processing a call to *tmpfile()*.

69300 An error message may be written to standard error if the stream cannot be opened.

69301 **RETURN VALUE**

69302 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is
 69303 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

69304 **ERRORS**69305 The *tmpfile()* function shall fail if:

69306 CX [EINTR] A signal was caught during *tmpfile()*.

69307 CX [EMFILE] All file descriptors available to the process are currently open.

69308 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process.

69309 CX [ENFILE] The maximum allowable number of files is currently open in the system.

69310 CX [ENOSPC] The directory or file system which would contain the new file cannot be
 69311 expanded.

69312 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly
 69313 in an object of type *off_t*.

69314 The *tmpfile()* function may fail if:

69315 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

69316 CX [ENOMEM] Insufficient storage space is available.

69317 **EXAMPLES**69318 **Creating a Temporary File**

69319 The following example creates a temporary file for update, and returns a pointer to a stream for
 69320 the created file in the *fp* variable.

69321 #include <stdio.h>

69322 ...

69323 FILE *fp;

69324 fp = tmpfile ();

69325 APPLICATION USAGE

69326 It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the
69327 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number
69328 simultaneously open other than this limit and any limit on the number of open file descriptors
69329 or streams ({OPEN_MAX}, {FOPEN_MAX}, {STREAM_MAX}).

69330 RATIONALE

69331 None.

69332 FUTURE DIRECTIONS

69333 None.

69334 SEE ALSO

69335 [Section 2.5](#) (on page 495), *fopen()*, *mkdtemp()*, *tmpnam()*, *unlink()*

69336 XBD <stdio.h>

69337 CHANGE HISTORY

69338 First released in Issue 1. Derived from Issue 1 of the SVID.

69339 Issue 5

69340 Large File Summit extensions are added.

69341 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
69342 in previous issues.

69343 Issue 6

69344 Extensions beyond the ISO C standard are marked.

69345 The following new requirements on POSIX implementations derive from alignment with the
69346 Single UNIX Specification:

69347 In the ERRORS section, the [Eoverflow] condition is added. This change is to support
69348 large files.

69349 The [EMFILE] optional error condition is added.

69350 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999
69351 standard.

69352 Issue 7

69353 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may
69354 restrict the permissions of the file created.

69355 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

69356 SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for
69357 {STREAM_MAX} streams open.

69358 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0668 [14] is applied.

69359 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0372 [678] is applied.

69360 **NAME**

69361 tmpnam — create a name for a temporary file

69362 **SYNOPSIS**

```
69363 OB #include <stdio.h>
69364 char *tmpnam(char *s);
```

69365 **DESCRIPTION**

69366 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69367 conflict between the requirements described here and the ISO C standard is unintentional. This
 69368 volume of POSIX.1-2017 defers to the ISO C standard.

69369 The *tmpnam()* function shall generate a string that is a valid pathname that does not name an
 69370 existing file. The function is potentially capable of generating {TMP_MAX} different strings, but
 69371 any or all of them may already be in use by existing files and thus not be suitable return values.

69372 The *tmpnam()* function generates a different string each time it is called from the same process,
 69373 up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is
 69374 implementation-defined.

69375 The implementation shall behave as if no function defined in this volume of POSIX.1-2017,
 69376 except *tmpnam()*, calls *tmpnam()*.

69377 CX The *tmpnam()* function need not be thread-safe if called with a NULL parameter.

69378 **RETURN VALUE**

69379 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can
 69380 be generated, the *tmpnam()* function shall return a null pointer.

69381 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and
 69382 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the
 69383 argument *s* is not a null pointer, it is presumed to point to an array of at least `L_tmpnam` **chars**;
 69384 *tmpnam()* shall write its result in that array and shall return the argument as its value.

69385 **ERRORS**

69386 No errors are defined.

69387 **EXAMPLES**69388 **Generating a Pathname**

69389 The following example generates a unique pathname and stores it in the array pointed to by *ptr*.

```
69390 #include <stdio.h>
69391 ...
69392 char pathname[L_tmpnam+1];
69393 char *ptr;
69394 ptr = tmpnam(pathname);
```

69395 **APPLICATION USAGE**

69396 This function only creates pathnames. It is the application's responsibility to create and remove
 69397 the files.

69398 Between the time a pathname is created and the file is opened, it is possible for some other
 69399 process to create a file with the same name. Applications may find *tmpfile()* more useful.

69400 Applications should use the *tmpfile()*, *mkstemp()*, or *mkdtemp()* functions instead of the
 69401 obsolescent *tmpnam()* function.

69402 **RATIONALE**

69403 None.

69404 **FUTURE DIRECTIONS**69405 The *tmpnam()* function may be removed in a future version.69406 **SEE ALSO**69407 *fopen()*, *open()*, *mkdtemp()*, *tmpnam()*, *tmpfile()*, *unlink()*

69408 XBD <stdio.h>

69409 **CHANGE HISTORY**

69410 First released in Issue 1. Derived from Issue 1 of the SVID.

69411 **Issue 5**

69412 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

69413 **Issue 6**

69414 Extensions beyond the ISO C standard are marked.

69415 The normative text is updated to avoid use of the term “must” for application requirements.

69416 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

69417 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/142 is applied, updating the
69418 DESCRIPTION to allow implementations of the *tmpnam()* function to call *tmpnam()*.69419 **Issue 7**69420 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *tmpnam()* function
69421 need not be thread-safe if called with a NULL parameter.69422 The *tmpnam()* function is marked obsolescent.69423 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0669 [291] and XSH/TC1-2008/0670
69424 [291,429] are applied.

69425 **NAME**

69426 toascii †translate an integer to a 7-bit ASCII character

69427 **SYNOPSIS**

```
69428 OB XSI #include <ctype.h>
69429 int toascii(int c);
```

69430 **DESCRIPTION**69431 The *toascii()* function shall convert its argument into a 7-bit ASCII character.69432 **RETURN VALUE**69433 The *toascii()* function shall return the value (*c* &0x7f).69434 **ERRORS**

69435 No errors are returned.

69436 **EXAMPLES**

69437 None.

69438 **APPLICATION USAGE**69439 The *toascii()* function cannot be used portably in a localized application.69440 **RATIONALE**

69441 None.

69442 **FUTURE DIRECTIONS**69443 The *toascii()* function may be removed in a future version.69444 **SEE ALSO**69445 [isascii\(\)](#)69446 XBD [<ctype.h>](#)69447 **CHANGE HISTORY**

69448 First released in Issue 1. Derived from Issue 1 of the SVID.

69449 **Issue 7**69450 The *toascii()* function is marked obsolescent.

69451 **NAME**

69452 tolower, tolower_l — transliterate uppercase characters to lowercase

69453 **SYNOPSIS**

```
69454       #include <ctype.h>
69455       int tolower(int c);
69456 CX     int tolower_l(int c, locale_t locale);
```

69457 **DESCRIPTION**

69458 CX For *tolower()*: The functionality described on this reference page is aligned with the ISO C
 69459 standard. Any conflict between the requirements described here and the ISO C standard is
 69460 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

69461 CX The *tolower()* and *tolower_l()* functions have as a domain a type **int**, the value of which is
 69462 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the
 69463 behavior is undefined. If the argument of *tolower()* or *tolower_l()* represents an uppercase letter,
 69464 and there exists a corresponding lowercase letter as defined by character type information in the
 69465 current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the
 69466 result shall be the corresponding lowercase letter. All other arguments in the domain are
 69467 returned unchanged.

69468 CX The behavior is undefined if the *locale* argument to *tolower_l()* is the special locale object
 69469 LC_GLOBAL_LOCALE or is not a valid locale object handle.

69470 **RETURN VALUE**

69471 CX Upon successful completion, the *tolower()* and *tolower_l()* functions shall return the lowercase
 69472 letter corresponding to the argument passed; otherwise, they shall return the argument
 69473 unchanged.

69474 **ERRORS**

69475 No errors are defined.

69476 **EXAMPLES**

69477 None.

69478 **APPLICATION USAGE**

69479 None.

69480 **RATIONALE**

69481 None.

69482 **FUTURE DIRECTIONS**

69483 None.

69484 **SEE ALSO**69485 [setlocale\(\)](#), [uselocale\(\)](#)69486 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)69487 **CHANGE HISTORY**

69488 First released in Issue 1. Derived from Issue 1 of the SVID.

69489 **Issue 6**

69490 Extensions beyond the ISO C standard are marked.

69491 **Issue 7**

69492 The *tolower_l()* function is added from The Open Group Technical Standard, 2006, Extended API
69493 Set Part 4.

69494 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0671 [283] and XSH/TC1-2008/0672
69495 [283] are applied.

69496 **NAME**

69497 toupper, toupper_l — transliterate lowercase characters to uppercase

69498 **SYNOPSIS**

```
69499            #include <ctype.h>
69500            int toupper(int c);
69501 CX         int toupper_l(int c, locale_t locale);
```

69502 **DESCRIPTION**

69503 CX For *toupper()*: The functionality described on this reference page is aligned with the ISO C
69504 standard. Any conflict between the requirements described here and the ISO C standard is
69505 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

69506 CX The *toupper()* and *toupper_l()* functions have as a domain a type **int**, the value of which is
69507 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the
69508 behavior is undefined.

69509 CX If the argument of *toupper()* or *toupper_l()* represents a lowercase letter, and there exists a
69510 CX corresponding uppercase letter as defined by character type information in the current locale or
69511 in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the
69512 corresponding uppercase letter.

69513 All other arguments in the domain are returned unchanged.

69514 CX The behavior is undefined if the *locale* argument to *toupper_l()* is the special locale object
69515 *LC_GLOBAL_LOCALE* or is not a valid locale object handle.

69516 **RETURN VALUE**

69517 CX Upon successful completion, *toupper()* and *toupper_l()* shall return the uppercase letter
69518 corresponding to the argument passed; otherwise, they shall return the argument unchanged.

69519 **ERRORS**

69520 No errors are defined.

69521 **EXAMPLES**

69522 None.

69523 **APPLICATION USAGE**

69524 None.

69525 **RATIONALE**

69526 None.

69527 **FUTURE DIRECTIONS**

69528 None.

69529 **SEE ALSO**

69530 [setlocale\(\)](#), [uselocale\(\)](#)

69531 XBD [Chapter 7](#) (on page 135), [<ctype.h>](#), [<locale.h>](#)

69532 **CHANGE HISTORY**

69533 First released in Issue 1. Derived from Issue 1 of the SVID.

69534 **Issue 6**

69535 Extensions beyond the ISO C standard are marked.

69536 **Issue 7**

69537 SD5-XSH-ERN-181 is applied, clarifying the RETURN VALUE section.

69538 The *toupper_1()* function is added from The Open Group Technical Standard, 2006, Extended API
69539 Set Part 4.

69540 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0673 [283] and XSH/TC1-2008/0674
69541 [283] are applied.

69542 **NAME**

69543 towctrans, towctrans_l †wide-character transliteration

69544 **SYNOPSIS**

69545 #include <wctype.h>

69546 wint_t towctrans(wint_t *wc*, wctrans_t *desc*);69547 CX wint_t towctrans_l(wint_t *wc*, wctrans_t *desc*,
69548 locale_t *locale*);69549 **DESCRIPTION**69550 CX For *towctrans()*: The functionality described on this reference page is aligned with the ISO C
69551 standard. Any conflict between the requirements described here and the ISO C standard is
69552 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.69553 CX The *towctrans()* and *towctrans_l()* functions shall transliterate the wide-character code *wc* using
69554 the mapping described by *desc*.69555 CX The current setting of the *LC_CTYPE* category in the current locale or in the locale represented
69556 CX by *locale*, respectively, should be the same as during the call to *wctrans()* or *wctrans_l()* that
69557 returned the value *desc*.69558 If the value of *desc* is invalid (that is, not obtained by a call to *wctrans()* or *desc* is invalidated by a
69559 subsequent call to *setlocale()* that has affected category *LC_CTYPE*), the result is unspecified.69560 CX If the value of *desc* is invalid (that is, not obtained by a call to *wctrans_l()* with the same locale
69561 object *locale*) the result is unspecified.69562 CX An application wishing to check for error situations should set *errno* to 0 before calling
69563 *towctrans()* or *towctrans_l()*.69564 If *errno* is non-zero on return, an error has occurred.69565 The behavior is undefined if the *locale* argument to *towctrans_l()* is the special locale object
69566 *LC_GLOBAL_LOCALE* or is not a valid locale object handle.69567 **RETURN VALUE**69568 CX If successful, the *towctrans()* and *towctrans_l()* functions shall return the mapped value of *wc*
69569 using the mapping described by *desc*. Otherwise, they shall return *wc* unchanged.69570 **ERRORS**

69571 These functions may fail if:

69572 CX [EINVAL] *desc* contains an invalid transliteration descriptor.69573 **EXAMPLES**

69574 None.

69575 **APPLICATION USAGE**69576 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the
69577 table below, the functions in the left column are equivalent to the functions in the right column.

69578	<code>towlower(<i>wc</i>)</code>	<code>towctrans(<i>wc</i>, wctrans("tolower"))</code>
69579	<code>towlower_l(<i>wc</i>, <i>locale</i>)</code>	<code>towctrans_l(<i>wc</i>, wctrans("tolower"), <i>locale</i>)</code>
69580	<code>toupper(<i>wc</i>)</code>	<code>towctrans(<i>wc</i>, wctrans("toupper"))</code>
69581	<code>toupper_l(<i>wc</i>, <i>locale</i>)</code>	<code>towctrans_l(<i>wc</i>, wctrans("toupper"), <i>locale</i>)</code>

69582 **RATIONALE**

69583 None.

69584 **FUTURE DIRECTIONS**

69585 None.

69586 **SEE ALSO**69587 *tolower()*, *towupper()*, *wctrans()*69588 XBD <**wctype.h**>69589 **CHANGE HISTORY**

69590 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

69591 **Issue 6**

69592 Extensions beyond the ISO C standard are marked.

69593 **Issue 7**69594 The *towctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended
69595 API Set Part 4.69596 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0675 [302], XSH/TC1-2008/0676 [283],
69597 and XSH/TC1-2008/0677 [283] are applied.

69598 **NAME**

69599 tolower, tolower_l — transliterate uppercase wide-character code to lowercase

69600 **SYNOPSIS**

69601 #include <wctype.h>

69602 wint_t tolower(wint_t wc);

69603 CX wint_t tolower_l(wint_t wc, locale_t locale);

69604 **DESCRIPTION**69605 CX For *tolower()*: The functionality described on this reference page is aligned with the ISO C
69606 standard. Any conflict between the requirements described here and the ISO C standard is
69607 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.69608 CX The *tolower()* and *tolower_l()* functions have as a domain a type **wint_t**, the value of which
69609 the application shall ensure is a character representable as a **wchar_t**, and a wide-character code
69610 corresponding to a valid character in the locale used by the function or the value of WEOF. If
69611 CX the argument has any other value, the behavior is undefined. If the argument of *tolower()* or
69612 *tolower_l()* represents an uppercase wide-character code, and there exists a corresponding
69613 CX lowercase wide-character code as defined by character type information in the current locale or
69614 in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the
69615 corresponding lowercase wide-character code. All other arguments in the domain are returned
69616 unchanged.69617 CX The behavior is undefined if the *locale* argument to *tolower_l()* is the special locale object
69618 *LC_GLOBAL_LOCALE* or is not a valid locale object handle.69619 **RETURN VALUE**69620 CX Upon successful completion, the *tolower()* and *tolower_l()* functions shall return the
69621 lowercase letter corresponding to the argument passed; otherwise, they shall return the
69622 argument unchanged.69623 **ERRORS**

69624 No errors are defined.

69625 **EXAMPLES**

69626 None.

69627 **APPLICATION USAGE**

69628 None.

69629 **RATIONALE**

69630 None.

69631 **FUTURE DIRECTIONS**

69632 None.

69633 **SEE ALSO**69634 [setlocale\(\)](#), [uselocale\(\)](#)69635 XBD [Chapter 7](#) (on page 135), [<locale.h>](#), [<wctype.h>](#)69636 **CHANGE HISTORY**

69637 First released in Issue 4.

69638 **Issue 5**

69639 The following change has been made in this version for alignment with
69640 ISO/IEC 9899:1990/Amendment 1:1995 (E):

69641 The SYNOPSIS has been changed to indicate that this function and associated data types
69642 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

69643 **Issue 6**

69644 The normative text is updated to avoid use of the term “must” for application requirements.

69645 **Issue 7**

69646 The `towlower_l()` function is added from The Open Group Technical Standard, 2006, Extended
69647 API Set Part 4.

69648 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0678 [302], XSH/TC1-2008/0679 [283],
69649 and XSH/TC1-2008/0680 [283] are applied.

69650 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0373 [685] is applied.

69651 **NAME**

69652 towupper, towupper_l — transliterate lowercase wide-character code to uppercase

69653 **SYNOPSIS**

69654 #include <wctype.h>

69655 wint_t towupper(wint_t wc);

69656 CX wint_t towupper_l(wint_t wc, locale_t locale);

69657 **DESCRIPTION**69658 CX For *towupper()*: The functionality described on this reference page is aligned with the ISO C
69659 standard. Any conflict between the requirements described here and the ISO C standard is
69660 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.69661 CX The *towupper()* and *towupper_l()* functions have as a domain a type **wint_t**, the value of which
69662 the application shall ensure is a character representable as a **wchar_t**, and a wide-character code
69663 corresponding to a valid character in the locale used by the function or the value of WEOF. If
69664 CX the argument has any other value, the behavior is undefined. If the argument of *towupper()* or
69665 *towupper_l()* represents a lowercase wide-character code, and there exists a corresponding
69666 CX uppercase wide-character code as defined by character type information in the current locale or
69667 in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the
69668 corresponding uppercase wide-character code. All other arguments in the domain are returned
69669 unchanged.69670 CX The behavior is undefined if the *locale* argument to *towupper_l()* is the special locale object
69671 *LC_GLOBAL_LOCALE* or is not a valid locale object handle.69672 **RETURN VALUE**69673 CX Upon successful completion, the *towupper()* and *towupper_l()* functions shall return the
69674 uppercase letter corresponding to the argument passed. Otherwise, they shall return the
69675 argument unchanged.69676 **ERRORS**

69677 No errors are defined.

69678 **EXAMPLES**

69679 None.

69680 **APPLICATION USAGE**

69681 None.

69682 **RATIONALE**

69683 None.

69684 **FUTURE DIRECTIONS**

69685 None.

69686 **SEE ALSO**69687 [setlocale\(\)](#), [uselocale\(\)](#)69688 XBD [Chapter 7](#) (on page 135), [<locale.h>](#), [<wctype.h>](#)69689 **CHANGE HISTORY**

69690 First released in Issue 4.

69691 **Issue 5**

69692 The following change has been made in this version for alignment with
69693 ISO/IEC 9899:1990/Amendment 1:1995 (E):

69694 The SYNOPSIS has been changed to indicate that this function and associated data types
69695 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

69696 **Issue 6**

69697 The normative text is updated to avoid use of the term “must” for application requirements.

69698 **Issue 7**

69699 The `towupper_l()` function is added from The Open Group Technical Standard, 2006, Extended
69700 API Set Part 4.

69701 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0681 [302], XSH/TC1-2008/0682 [283],
69702 and XSH/TC1-2008/0683 [283] are applied.

69703 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0374 [685] is applied.

69704 **NAME**

69705 trunc, truncf, trunc — round to truncated integer value

69706 **SYNOPSIS**

```
69707 #include <math.h>
69708 double trunc(double x);
69709 float truncf(float x);
69710 long double trunc1(long double x);
```

69711 **DESCRIPTION**

69712 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69713 conflict between the requirements described here and the ISO C standard is unintentional. This
 69714 volume of POSIX.1-2017 defers to the ISO C standard.

69715 These functions shall round their argument to the integer value, in floating format, nearest to but
 69716 no larger in magnitude than the argument.

69717 **RETURN VALUE**

69718 Upon successful completion, these functions shall return the truncated integer value.

69719 MX The result shall have the same sign as x .69720 MX If x is NaN, a NaN shall be returned.69721 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.69722 **ERRORS**

69723 No errors are defined.

69724 **EXAMPLES**

69725 None.

69726 **APPLICATION USAGE**

69727 The integral value returned by these functions need not be expressible as an `intmax_t`. The
 69728 return value should be tested before assigning it to an integer type to avoid the undefined
 69729 results of an integer overflow.

69730 These functions may raise the inexact floating-point exception if the result differs in value from
 69731 the argument.

69732 **RATIONALE**

69733 None.

69734 **FUTURE DIRECTIONS**

69735 None.

69736 **SEE ALSO**69737 XBD [<math.h>](#)69738 **CHANGE HISTORY**

69739 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

69740 **Issue 7**

69741 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0684 [346] is applied.

69742 **NAME**

69743 truncate — truncate a file to a specified length

69744 **SYNOPSIS**

69745 #include <unistd.h>

69746 int truncate(const char *path, off_t length);

69747 **DESCRIPTION**69748 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be
69749 equal to *length* bytes.69750 If the file previously was larger than *length*, the extra data is discarded. If the file was previously
69751 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

69752 The application shall ensure that the process has write permission for the file.

69753 XSI If the request would cause the file size to exceed the soft file size limit for the process, the
69754 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.69755 The *truncate()* function shall not modify the file offset for any open file descriptions associated
69756 with the file. Upon successful completion, *truncate()* shall mark for update the last data
69757 modification and last file status change timestamps of the file, and the S_ISUID and S_ISGID bits
69758 of the file mode may be cleared.69759 **RETURN VALUE**69760 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*
69761 set to indicate the error.69762 **ERRORS**69763 The *truncate()* function shall fail if:

69764 [EINTR] A signal was caught during execution.

69765 [EINVAL] The *length* argument was less than 0.

69766 [EFBIG] or [EINVAL]

69767 The *length* argument was greater than the maximum file size.

69768 [EIO] An I/O error occurred while reading from or writing to a file system.

69769 [EACCES] A component of the path prefix denies search permission, or write permission
69770 is denied on the file.

69771 [EISDIR] The named file is a directory.

69772 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
69773 argument.

69774 [ENAMETOOLONG]

69775 The length of a component of a pathname is longer than {NAME_MAX}.

69776 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.69777 [ENOTDIR] A component of the path prefix names an existing file that is neither a
69778 directory nor a symbolic link to a directory, or the *path* argument contains at
69779 least one non-*<slash>* character and ends with one or more trailing *<slash>*
69780 characters and the last pathname component names an existing file that is
69781 neither a directory nor a symbolic link to a directory.

- 69782 [EROFS] The named file resides on a read-only file system.
- 69783 The *truncate()* function may fail if:
- 69784 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
69785 resolution of the *path* argument.
- 69786 [ENAMETOOLONG]
69787 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
69788 symbolic link produced an intermediate result with a length that exceeds
69789 {PATH_MAX}.
- 69790 **EXAMPLES**
69791 None.
- 69792 **APPLICATION USAGE**
69793 None.
- 69794 **RATIONALE**
69795 None.
- 69796 **FUTURE DIRECTIONS**
69797 None.
- 69798 **SEE ALSO**
69799 [open\(\)](#)
69800 XBD [<unistd.h>](#)
- 69801 **CHANGE HISTORY**
69802 First released in Issue 4, Version 2.
- 69803 **Issue 5**
69804 Moved from X/OPEN UNIX extension to BASE.
69805 Large File Summit extensions are added.
- 69806 **Issue 6**
69807 This reference page is split out from the *ftruncate()* reference page.
69808 The normative text is updated to avoid use of the term “must” for application requirements.
69809 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
69810 [ELOOP] error condition is added.
- 69811 **Issue 7**
69812 Austin Group Interpretation 1003.1-2001 #143 is applied.
69813 The *truncate()* function is moved from the XSI option to the Base.
69814 Changes are made related to support for finegrained timestamps.
69815 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
69816 pathname exists but is not a directory or a symbolic link to a directory.
69817 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0685 [324] is applied.
69818 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0375 [489] is applied.

69819 **NAME**
69820 truncf, trunc — round to truncated integer value

69821 **SYNOPSIS**
69822 #include <math.h>

69823 float truncf(float x);
69824 long double trunc1(long double x);

69825 **DESCRIPTION**
69826 Refer to *trunc()*.

69827 **NAME**

69828 tsearch — search a binary search tree

69829 **SYNOPSIS**

```
69830 XSI #include <search.h>
69831 void *tsearch(const void *key, void **rootp,
69832 int (*compar)(const void *, const void *));
```

69833 **DESCRIPTION**69834 Refer to [tdelete\(\)](#).

69835 **NAME**

69836 `ttyname, ttyname_r` ¶ find the pathname of a terminal

69837 **SYNOPSIS**

69838 `#include <unistd.h>`

69839 `char *ttyname(int fildev);`

69840 `int ttyname_r(int fildev, char *name, size_t namesize);`

69841 **DESCRIPTION**

69842 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname
 69843 of the terminal associated with file descriptor *fildev*. The application shall not modify the string
 69844 returned. The returned pointer might be invalidated or the string content might be overwritten
 69845 by a subsequent call to `ttyname()`. The returned pointer and the string content might also be
 69846 invalidated if the calling thread is terminated.

69847 The `ttyname()` function need not be thread-safe.

69848 The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated
 69849 with the file descriptor *fildev* in the character array referenced by *name*. The array is *namesize*
 69850 characters long and should have space for the name and the terminating null character. The
 69851 maximum length of the terminal name shall be {TTY_NAME_MAX}.

69852 **RETURN VALUE**

69853 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null
 69854 pointer shall be returned and *errno* set to indicate the error.

69855 If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be
 69856 returned to indicate the error.

69857 **ERRORS**

69858 The `ttyname()` function may fail if:

69859 [EBADF] The *fildev* argument is not a valid file descriptor.

69860 [ENOTTY] The file associated with the *fildev* argument is not a terminal.

69861 The `ttyname_r()` function may fail if:

69862 [EBADF] The *fildev* argument is not a valid file descriptor.

69863 [ENOTTY] The file associated with the *fildev* argument is not a terminal.

69864 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
 69865 including the terminating null character.

69866 **EXAMPLES**

69867 None.

69868 **APPLICATION USAGE**

69869 None.

69870 **RATIONALE**

69871 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a
 69872 reference to an undefined term.

69873 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-
 69874 zero value if it fails. The non-thread-safe version may return the name in a static data area that
 69875 may be overwritten by each call.

69876 **FUTURE DIRECTIONS**

69877 None.

69878 **SEE ALSO**69879 XBD <[unistd.h](#)>69880 **CHANGE HISTORY**

69881 First released in Issue 1. Derived from Issue 1 of the SVID.

69882 **Issue 5**69883 The `ttyname_r()` function is included for alignment with the POSIX Threads Extension.69884 A note indicating that the `ttyname()` function need not be reentrant is added to the
69885 DESCRIPTION.69886 **Issue 6**69887 The `ttyname_r()` function is marked as part of the Thread-Safe Functions option.69888 The following new requirements on POSIX implementations derive from alignment with the
69889 Single UNIX Specification:69890 The statement that `errno` is set on error is added.

69891 The [EBADF] and [ENOTTY] optional error conditions are added.

69892 **Issue 7**

69893 Austin Group Interpretation 1003.1-2001 #156 is applied.

69894 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

69895 The `ttyname_r()` function is moved from the Thread-Safe Functions option to the Base.

69896 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0686 [75] is applied.

69897 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0376 [656] is applied.

69898 **NAME**

69899 twalk — traverse a binary search tree

69900 **SYNOPSIS**

```
69901 XSI #include <search.h>
69902 void twalk(const void *root,
69903           void (*action)(const void *, VISIT, int ));
```

69904 **DESCRIPTION**

69905 Refer to *tdelete()*.

69906 **NAME**

69907 daylight, timezone, tzname, tzset ‡set timezone conversion information

69908 **SYNOPSIS**

69909 #include <time.h>

```

69910 XSI extern int daylight;
69911      extern long timezone;
69912 CX   extern char *tzname[2];
69913      void tzset(void);

```

69914 **DESCRIPTION**

69915 The `tzset()` function shall use the value of the environment variable `TZ` to set time conversion information used by `ctime()`, `localtime()`, `mktime()`, and `strptime()`. If `TZ` is absent from the environment, implementation-defined default timezone information shall be used.

69918 The `tzset()` function shall set the external variable `tzname` as follows:

```

69919      tzname[0] = "std";
69920      tzname[1] = "dst";

```

69921 where `std` and `dst` are as described in XBD Chapter 8 (on page 173).

69922 XSI The `tzset()` function also shall set the external variable `daylight` to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable `timezone` shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

69926 XSI If a thread accesses `tzname`, `daylight`, or `timezone` directly while another thread is in a call to `tzset()`, or to any function that is required or allowed to set timezone information as if by calling `tzset()`, the behavior is undefined.

69929 **RETURN VALUE**

69930 The `tzset()` function shall not return a value.

69931 **ERRORS**

69932 No errors are defined.

69933 **EXAMPLES**

69934 Example `TZ` variables and their timezone differences are given in the table below:

	<i>TZ</i>	<i>timezone</i>
69935		
69936	EST5EDT	5*60*60
69937	GMT0	0*60*60
69938	JST-9	-9*60*60
69939	MET-1MEST	-1*60*60
69940	MST7MDT	7*60*60
69941	PST8PDT	8*60*60

69942 **APPLICATION USAGE**

69943 Since the `ctime()`, `localtime()`, `mktime()`, `strptime()`, and `strptime_l()` functions are required to set timezone information as if by calling `tzset()`, there is no need for an explicit `tzset()` call before using these functions. However, portable applications should call `tzset()` explicitly before using `ctime_r()` or `localtime_r()` because setting timezone information is optional for those functions.

69947 **RATIONALE**

69948 None.

69949 **FUTURE DIRECTIONS**

69950 None.

69951 **SEE ALSO**69952 *ctime()*, *localtime()*, *mktime()*, *strftime()*69953 XBD Chapter 8 (on page 173), [<time.h>](#)69954 **CHANGE HISTORY**

69955 First released in Issue 1. Derived from Issue 1 of the SVID.

69956 **Issue 6**

69957 The example is corrected.

69958 **Issue 7**

69959 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0377 [880] is applied.

69960 **NAME**

69961 ulimit — get and set process limits

69962 **SYNOPSIS**

```
69963 OB XSI #include <ulimit.h>
69964 long ulimit(int cmd, ...);
```

69965 **DESCRIPTION**

69966 The *ulimit()* function shall control process limits. The process limits that can be controlled by
 69967 this function include the maximum size of a single file that can be written (this is equivalent to
 69968 using *setrlimit()* with `RLIMIT_FSIZE`). The *cmd* values, defined in `<ulimit.h>`, include:

69969 **UL_GETFSIZE** Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit shall be in
 69970 units of 512-byte blocks and shall be inherited by child processes. Files of any
 69971 size can be read. The return value shall be the integer part of the soft file size
 69972 limit divided by 512. If the result cannot be represented as a **long**, the result is
 69973 unspecified.

69974 **UL_SETFSIZE** Set the file size limit for output operations of the process to the value of the
 69975 second argument, taken as a **long**, multiplied by 512. If the result would
 69976 overflow an **rlim_t**, the actual value set is unspecified. Any process may
 69977 decrease its own limit, but only a process with appropriate privileges may
 69978 increase the limit. The return value shall be the integer part of the new file size
 69979 limit divided by 512.

69980 The *ulimit()* function shall not change the setting of *errno* if successful.

69981 As all return values are permissible in a successful situation, an application wishing to check for
 69982 error situations should set *errno* to 0, then call *ulimit()*, and, if it returns `-1`, check to see if *errno* is
 69983 non-zero.

69984 **RETURN VALUE**

69985 Upon successful completion, *ulimit()* shall return the value of the requested limit. Otherwise, `-1`
 69986 shall be returned and *errno* set to indicate the error.

69987 **ERRORS**

69988 The *ulimit()* function shall fail and the limit shall be unchanged if:

69989 `[EINVAL]` The *cmd* argument is not valid.

69990 `[EPERM]` A process not having appropriate privileges attempts to increase its file size
 69991 limit.

69992 **EXAMPLES**

69993 None.

69994 **APPLICATION USAGE**

69995 Since the *ulimit()* function uses type **long** rather than **rlim_t**, this function is not sufficient for file
 69996 sizes on many current systems. Applications should use the *getrlimit()* or *setrlimit()* functions
 69997 instead of the obsolescent *ulimit()* function.

69998 **RATIONALE**

69999 None.

70000 FUTURE DIRECTIONS

70001 The *ulimit()* function may be removed in a future version.

70002 SEE ALSO

70003 *exec*, *getrlimit()*, *write()*

70004 XBD <**ulimit.h**>

70005 CHANGE HISTORY

70006 First released in Issue 1. Derived from Issue 1 of the SVID.

70007 Issue 5

70008 In the description of `UL_SETFSIZE`, the text is corrected to refer to `rlim_t` rather than the
70009 spurious `rlimit_t`.

70010 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

70011 Issue 7

70012 The *ulimit()* function is marked obsolescent.

70013 **NAME**

70014 umask — set and get the file mode creation mask

70015 **SYNOPSIS**

70016 #include <sys/stat.h>

70017 mode_t umask(mode_t *cmask*);70018 **DESCRIPTION**

70019 The *umask()* function shall set the file mode creation mask of the process to *cmask* and return the
 70020 previous value of the mask. Only the file permission bits of *cmask* (see <sys/stat.h>) are used; the
 70021 meaning of the other bits is implementation-defined.

70022 The file mode creation mask of the process is used to turn off permission bits in the *mode*
 70023 argument supplied during calls to the following functions:

70024 *open()*, *openat()*, *creat()*, *mkdir()*, *mkdirat()*, *mkfifo()*, and *mkfifoat()*70025 XSI *mknod()*, *mknodat()*70026 MSG *mq_open()*70027 *sem_open()*70028 Bit positions that are set in *cmask* are cleared in the mode of the created file.70029 **RETURN VALUE**

70030 The file permission bits in the value returned by *umask()* shall be the previous value of the file
 70031 mode creation mask. The state of any other bits in that value is unspecified, except that a
 70032 subsequent call to *umask()* with the returned value as *cmask* shall leave the state of the mask the
 70033 same as its state before the first call, including any unspecified use of those bits.

70034 **ERRORS**

70035 No errors are defined.

70036 **EXAMPLES**

70037 None.

70038 **APPLICATION USAGE**

70039 None.

70040 **RATIONALE**

70041 Unsigned argument and return types for *umask()* were proposed. The return type and the
 70042 argument were both changed to **mode_t**.

70043 Historical implementations have made use of additional bits in *cmask* for their implementation-
 70044 defined purposes. The addition of the text that the meaning of other bits of the field is
 70045 implementation-defined permits these implementations to conform to this volume of
 70046 POSIX.1-2017.

70047 **FUTURE DIRECTIONS**

70048 None.

70049 **SEE ALSO**70050 *creat()*, *exec*, *mkdir()*, *mkfifo()*, *mknod()*, *mq_open()*, *open()*, *sem_open()*

70051 XBD <sys/stat.h>, <sys/types.h>

70052 **CHANGE HISTORY**

70053 First released in Issue 1. Derived from Issue 1 of the SVID.

70054 **Issue 6**

70055 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

70056 The following new requirements on POSIX implementations derive from alignment with the
70057 Single UNIX Specification:

70058 The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
70059 required for conforming implementations of previous POSIX specifications, it was not
70060 required for UNIX applications.

70061 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/143 is applied, adding the `mknod()`,
70062 `mq_open()`, and `sem_open()` functions to the DESCRIPTION and SEE ALSO sections.

70063 **NAME**

70064 uname — get the name of the current system

70065 **SYNOPSIS**70066 #include <sys/utsname.h>
70067 int uname(struct utsname *name);70068 **DESCRIPTION**70069 The *uname()* function shall store information identifying the current system in the structure
70070 pointed to by *name*.70071 The *uname()* function uses the **utsname** structure defined in <sys/utsname.h>.70072 The *uname()* function shall return a string naming the current system in the character array
70073 *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-
70074 defined communications network. The arrays *release* and *version* shall further identify the
70075 operating system. The array *machine* shall contain a name that identifies the hardware that the
70076 system is running on.

70077 The format of each member is implementation-defined.

70078 **RETURN VALUE**70079 Upon successful completion, a non-negative value shall be returned. Otherwise, *-1* shall be
70080 returned and *errno* set to indicate the error.70081 **ERRORS**

70082 No errors are defined.

70083 **EXAMPLES**

70084 None.

70085 **APPLICATION USAGE**70086 The inclusion of the *nodename* member in this structure does not imply that it is sufficient
70087 information for interfacing to communications networks.70088 **RATIONALE**70089 The values of the structure members are not constrained to have any relation to the version of
70090 this volume of POSIX.1-2017 implemented in the operating system. An application should
70091 instead depend on *_POSIX_VERSION* and related constants defined in <unistd.h>.70092 This volume of POSIX.1-2017 does not define the sizes of the members of the structure and
70093 permits them to be of different sizes, although most implementations define them all to be the
70094 same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not
70095 enough for use with many networks.70096 The *uname()* function originated in System III, System V, and related implementations, and it
70097 does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in
70098 those historical implementations.70099 4.3 BSD has *gethostname()* and *gethostid()*, which return a symbolic name and a numeric value,
70100 respectively. There are related *sethostname()* and *sethostid()* functions that are used to set the
70101 values the other two functions return. The former functions are included in this specification, the
70102 latter are not.70103 **FUTURE DIRECTIONS**

70104 None.

70105 **SEE ALSO**

70106 XBD <[sys/utsname.h](#)>

70107 **CHANGE HISTORY**

70108 First released in Issue 1. Derived from Issue 1 of the SVID.

70109 **NAME**

70110 ungetc — push byte back into input stream

70111 **SYNOPSIS**

70112 #include <stdio.h>

70113 int ungetc(int *c*, FILE **stream*);70114 **DESCRIPTION**

70115 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 70116 conflict between the requirements described here and the ISO C standard is unintentional. This
 70117 volume of POSIX.1-2017 defers to the ISO C standard.

70118 The *ungetc()* function shall push the byte specified by *c* (converted to an **unsigned char**) back
 70119 onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by
 70120 subsequent reads on that stream in the reverse order of their pushing. A successful intervening
 70121 CX call (with the stream pointed to by *stream*) to a file-positioning function (*fseek()*, *fseeko()*,
 70122 CX *fsetpos()*, or *rewind()*) or *fflush()* shall discard any pushed-back bytes for the stream. The
 70123 external storage corresponding to the stream shall be unchanged.

70124 One byte of push-back shall be provided. If *ungetc()* is called too many times on the same stream
 70125 without an intervening read or file-positioning operation on that stream, the operation may fail.

70126 If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall
 70127 be left unchanged.

70128 A successful call to *ungetc()* shall clear the end-of-file indicator for the stream. The value of the
 70129 file-position indicator for the stream after all pushed-back bytes have been read, or discarded by
 70130 CX calling *fseek()*, *fseeko()*, *fsetpos()*, or *rewind()* (but not *fflush()*), shall be the same as it was before
 70131 the bytes were pushed back. The file-position indicator is decremented by each successful call to
 70132 *ungetc()*; if its value was 0 before a call, its value is unspecified after the call.

70133 **RETURN VALUE**

70134 Upon successful completion, *ungetc()* shall return the byte pushed back after conversion.
 70135 Otherwise, it shall return EOF.

70136 **ERRORS**

70137 No errors are defined.

70138 **EXAMPLES**

70139 None.

70140 **APPLICATION USAGE**

70141 None.

70142 **RATIONALE**

70143 None.

70144 **FUTURE DIRECTIONS**

70145 None.

70146 **SEE ALSO**70147 [Section 2.5](#) (on page 495), *fseek()*, *getc()*, *fsetpos()*, *read()*, *rewind()*, *setbuf()*70148 XBD [<stdio.h>](#)70149 **CHANGE HISTORY**

70150 First released in Issue 1. Derived from Issue 1 of the SVID.

70151 **Issue 7**70152 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0687 [87,93], XSH/TC1-2008/0688
70153 [87], and XSH/TC1-2008/0689 [14] are applied.

70154 **NAME**

70155 ungetwc — push wide-character code back into the input stream

70156 **SYNOPSIS**

70157 #include <stdio.h>

70158 #include <wchar.h>

70159 wint_t ungetwc(wint_t *wc*, FILE **stream*);70160 **DESCRIPTION**

70161 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 70162 conflict between the requirements described here and the ISO C standard is unintentional. This
 70163 volume of POSIX.1-2017 defers to the ISO C standard.

70164 The *ungetwc()* function shall push the character corresponding to the wide-character code
 70165 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters
 70166 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A
 70167 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function
 70168 CX (*fseek()*, *fseeko()*, *fsetpos()*, or *rewind()*) or *fflush()* shall discard any pushed-back characters for
 70169 the stream. The external storage corresponding to the stream is unchanged.

70170 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on
 70171 the same stream without an intervening read or file-positioning operation on that stream, the
 70172 operation may fail.

70173 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream
 70174 shall be left unchanged.

70175 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the
 70176 file-position indicator for the stream after all pushed-back characters have been read, or
 70177 CX discarded by calling *fseek()*, *fseeko()*, *fsetpos()*, or *rewind()* (but not *fflush()*), shall be the same as
 70178 it was before the characters were pushed back. The file-position indicator is decremented (by
 70179 one or more) by each successful call to *ungetwc()*; if its value was 0 before a call, its value is
 70180 unspecified after the call.

70181 **RETURN VALUE**

70182 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to
 70183 the pushed-back character. Otherwise, it shall return WEOF.

70184 **ERRORS**

70185 The *ungetwc()* function may fail if:

70186 CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not
 70187 correspond to a valid character.

70188 **EXAMPLES**

70189 None.

70190 **APPLICATION USAGE**

70191 None.

70192 **RATIONALE**

70193 None.

70194 **FUTURE DIRECTIONS**

70195 None.

70196 **SEE ALSO**70197 [Section 2.5](#) (on page 495), [fseek\(\)](#), [fsetpos\(\)](#), [read\(\)](#), [rewind\(\)](#), [setbuf\(\)](#)70198 XBD [<stdio.h>](#), [<wchar.h>](#)70199 **CHANGE HISTORY**

70200 First released in Issue 4. Derived from the MSE working draft.

70201 **Issue 5**70202 The Optional Header (OH) marking is removed from [<stdio.h>](#).70203 **Issue 6**

70204 The [EILSEQ] optional error condition is marked CX.

70205 **Issue 7**70206 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0690 [87,93], XSH/TC1-2008/0691
70207 [87], and XSH/TC1-2008/0692 [14] are applied.

70208 **NAME**

70209 unlink, unlinkat — remove a directory entry

70210 **SYNOPSIS**

70211 #include <unistd.h>

70212 int unlink(const char *path);

70213 OH #include <fcntl.h>

70214 int unlinkat(int fd, const char *path, int flag);

70215 **DESCRIPTION**

70216 The *unlink()* function shall remove a link to a file. If *path* names a symbolic link, *unlink()* shall
 70217 remove the symbolic link named by *path* and shall not affect any file or directory named by the
 70218 contents of the symbolic link. Otherwise, *unlink()* shall remove the link named by the pathname
 70219 pointed to by *path* and shall decrement the link count of the file referenced by the link.

70220 When the file's link count becomes 0 and no process has the file open, the space occupied by the
 70221 file shall be freed and the file shall no longer be accessible. If one or more processes have the file
 70222 open when the last link is removed, the link shall be removed before *unlink()* returns, but the
 70223 removal of the file contents shall be postponed until all references to the file are closed.

70224 The *path* argument shall not name a directory unless the process has appropriate privileges and
 70225 the implementation supports using *unlink()* on directories.

70226 Upon successful completion, *unlink()* shall mark for update the last data modification and last
 70227 file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last
 70228 file status change timestamp of the file shall be marked for update.

70229 The *unlinkat()* function shall be equivalent to the *unlink()* or *rmdir()* function except in the case
 70230 where *path* specifies a relative path. In this case the directory entry to be removed is determined
 70231 relative to the directory associated with the file descriptor *fd* instead of the current working
 70232 directory. If the access mode of the open file description associated with the file descriptor is not
 70233 O_SEARCH, the function shall check whether directory searches are permitted using the current
 70234 permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the
 70235 function shall not perform the check.

70236 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 70237 in <fcntl.h>:

70238 AT_REMOVEDIR

70239 Remove the directory entry specified by *fd* and *path* as a directory, not a normal file.

70240 If *unlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 70241 directory shall be used and the behavior shall be identical to a call to *unlink()* or *rmdir()*
 70242 respectively, depending on whether or not the AT_REMOVEDIR bit is set in *flag*.

70243 **RETURN VALUE**

70244 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 70245 return -1 and set *errno* to indicate the error. If -1 is returned, the named file shall not be changed.

70246 **ERRORS**

70247 These functions shall fail and shall not unlink the file if:

70248 [EACCES] Search permission is denied for a component of the path prefix, or write
 70249 permission is denied on the directory containing the directory entry to be
 70250 removed.

70251	[EBUSY]	The file named by the <i>path</i> argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error.
70252		
70253		
70254	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
70255		
70256	[ENAMETOOLONG]	
70257		The length of a component of a pathname is longer than {NAME_MAX}.
70258	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
70259	[ENOTDIR]	A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the <i>path</i> argument contains at least one non- <code><slash></code> character and ends with one or more trailing <code><slash></code> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.
70260		
70261		
70262		
70263		
70264	[EPERM]	The file named by <i>path</i> is a directory, and either the calling process does not have appropriate privileges, or the implementation prohibits using <i>unlink()</i> on directories.
70265		
70266		
70267	XSI [EPERM] or [EACCES]	
70268		The S_ISVTX flag is set on the directory containing the file referred to by the <i>path</i> argument and the process does not satisfy the criteria specified in XBD Section 4.3 (on page 108).
70269		
70270		
70271	[EROFS]	The directory entry to be unlinked is part of a read-only file system.
70272		The <i>unlinkat()</i> function shall fail if:
70273	[EACCES]	The access mode of the open file description associated with <i>fd</i> is not O_SEARCH and the permissions of the directory underlying <i>fd</i> do not permit directory searches.
70274		
70275		
70276	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.
70277		
70278	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is a file descriptor associated with a non-directory file.
70279		
70280	[EEXIST] or [ENOTEMPTY]	
70281		The <i>flag</i> parameter has the AT_REMOVEDIR bit set and the <i>path</i> argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.
70282		
70283		
70284	[ENOTDIR]	The <i>flag</i> parameter has the AT_REMOVEDIR bit set and <i>path</i> does not name a directory.
70285		
70286		These functions may fail and not unlink the file if:
70287	XSI [EBUSY]	The file named by <i>path</i> is a named STREAM.
70288	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
70289		
70290	[ENAMETOOLONG]	
70291		The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.
70292		
70293		

70294 [ETXTBSY] The entry to be unlinked is the last directory entry to a pure procedure (shared
70295 text) file that is being executed.

70296 The *unlinkat()* function may fail if:

70297 [EINVAL] The value of the *flag* argument is not valid.

70298 EXAMPLES

70299 Removing a Link to a File

70300 The following example shows how to remove a link to a file named `/home/cnd/mod1` by
70301 removing the entry named `/modules/pass1`.

```
70302 #include <unistd.h>
70303
70304 char *path = "/modules/pass1";
70305 int status;
70306 ...
70307 status = unlink(path);
```

70307 Checking for an Error

70308 The following example fragment creates a temporary password lock file named `LOCKFILE`,
70309 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for
70310 writing, *unlink()* is used to remove the link between the file descriptor and `LOCKFILE`.

```
70311 #include <sys/types.h>
70312 #include <stdio.h>
70313 #include <fcntl.h>
70314 #include <errno.h>
70315 #include <unistd.h>
70316 #include <sys/stat.h>
70317
70318 #define LOCKFILE "/etc/ptmp"
70319
70320 int pfd; /* Integer for file descriptor returned by open call. */
70321 FILE *fpfd; /* File pointer for use in putpwent(). */
70322 ...
70323 /* Open password Lock file. If it exists, this is an error. */
70324 if ((pfd = open(LOCKFILE, O_WRONLY|O_CREAT|O_EXCL, S_IRUSR
70325 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
70326     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
70327     exit(1);
70328 }
70329
70330 /* Lock file created; proceed with fdopen of lock file so that
70331 putpwent() can be used.
70332 */
70333 if ((fpfd = fdopen(pfd, "w")) == NULL) {
70334     close(pfd);
70335     unlink(LOCKFILE);
70336     exit(1);
70337 }
```

70335 **Replacing Files**

70336 The following example fragment uses *unlink()* to discard links to files, so that they can be
 70337 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error
 70338 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can
 70339 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```

70340 #include <sys/types.h>
70341 #include <stdio.h>
70342 #include <fcntl.h>
70343 #include <errno.h>
70344 #include <unistd.h>
70345 #include <sys/stat.h>

70346 #define LOCKFILE "/etc/ptmp"
70347 #define PASSWDFILE "/etc/passwd"
70348 #define SAVEFILE "/etc/opasswd"
70349 ...
70350 /* If no change was made, assume error and leave passwd unchanged. */
70351 if (!valid_change) {
70352     fprintf(stderr, "Could not change password for user %s\n", user);
70353     unlink(LOCKFILE);
70354     exit(1);
70355 }

70356 /* Change permissions on new password file. */
70357 chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

70358 /* Remove saved password file. */
70359 unlink(SAVEFILE);

70360 /* Save current password file. */
70361 link(PASSWDFILE, SAVEFILE);

70362 /* Remove current password file. */
70363 unlink(PASSWDFILE);

70364 /* Save new password file as current password file. */
70365 link(LOCKFILE, PASSWDFILE);

70366 /* Remove lock file. */
70367 unlink(LOCKFILE);

70368 exit(0);

```

70369 **APPLICATION USAGE**

70370 Applications should use *rmdir()* to remove a directory.

70371 **RATIONALE**

70372 Unlinking a directory is restricted to the superuser in many historical implementations for
 70373 reasons given in *link()* (see also *rename()*).

70374 The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume
 70375 of POSIX.1-2017 does not cover the system administration concepts of mounting and
 70376 unmounting, the description of the error was changed to “resource busy”. (This meaning is used
 70377 by some device drivers when a second process tries to open an exclusive use device.) The
 70378 wording is also intended to allow implementations to refuse to remove a directory if it is the root
 70379 or current working directory of any process.

70380 The standard developers reviewed TR 24715-2006 and noted that LSB-conforming
 70381 implementations may return [EISDIR] instead of [EPERM] when unlinking a directory. A change
 70382 to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was
 70383 considered, but decided against since it would break existing strictly conforming and
 70384 conforming applications. Applications written for portability to both POSIX.1-2017 and the LSB
 70385 should be prepared to handle either error code.

70386 The purpose of the *unlinkat()* function is to remove directory entries in directories other than the
 70387 current working directory without exposure to race conditions. Any part of the path of a file
 70388 could be changed in parallel to a call to *unlink()*, resulting in unspecified behavior. By opening a
 70389 file descriptor for the target directory and using the *unlinkat()* function it can be guaranteed that
 70390 the removed directory entry is located relative to the desired directory.

70391 **FUTURE DIRECTIONS**

70392 None.

70393 **SEE ALSO**

70394 *close()*, *link()*, *remove()*, *rename()*, *rmdir()*, *symlink()*

70395 XBD Section 4.3 (on page 108), *<fcntl.h>*, *<unistd.h>*

70396 **CHANGE HISTORY**

70397 First released in Issue 1. Derived from Issue 1 of the SVID.

70398 **Issue 5**

70399 The [EBUSY] error is added to the optional part of the ERRORS section.

70400 **Issue 6**

70401 The following new requirements on POSIX implementations derive from alignment with the
 70402 Single UNIX Specification:

70403 In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.

70404 The [ELOOP] mandatory error condition is added.

70405 A second [ENAMETOOLONG] is added as an optional error condition.

70406 The [ETXTBSY] optional error condition is added.

70407 The following changes were made to align with the IEEE P1003.1a draft standard:

70408 The [ELOOP] optional error condition is added.

70409 The normative text is updated to avoid use of the term “must” for application requirements.

70410 **Issue 7**

70411 Austin Group Interpretation 1003.1-2001 #143 is applied.

70412 Austin Group Interpretation 1003.1-2001 #181 is applied, updating the requirements for
 70413 operations when the S_ISVTX bit is set.

70414 Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM]
 70415 and [EISDIR].

70416 The *unlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API
 70417 Set Part 2.

70418 Changes are made related to support for finegrained timestamps.

70419 Changes are made to allow a directory to be opened for searching.

70420 The [ENOTDIR] error condition is clarified to cover the condition where the last component of a
70421 pathname exists but is not a directory or a symbolic link to a directory.

70422 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0693 [461], XSH/TC1-2008/0694 [324],
70423 XSH/TC1-2008/0695 [278], and XSH/TC1-2008/0696 [278] are applied.

70424 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0378 [873], XSH/TC2-2008/0379 [591],
70425 XSH/TC2-2008/0380 [817], and XSH/TC2-2008/0381 [817] are applied.

70426 **NAME**

70427 unlockpt ‡'unlock a pseudo-terminal master/slave pair

70428 **SYNOPSIS**

```
70429 XSI       #include <stdlib.h>
70430       int unlockpt(int fildev);
```

70431 **DESCRIPTION**

70432 The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the master
70433 to which *fildev* refers.

70434 Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a
70435 pseudo-terminal device.

70436 **RETURN VALUE**

70437 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno*
70438 to indicate the error.

70439 **ERRORS**

70440 The *unlockpt()* function may fail if:

70441 [EBADF] The *fildev* argument is not a file descriptor open for writing.

70442 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.

70443 **EXAMPLES**

70444 None.

70445 **APPLICATION USAGE**

70446 None.

70447 **RATIONALE**

70448 See the RATIONALE section for *posix_openpt()*.

70449 **FUTURE DIRECTIONS**

70450 None.

70451 **SEE ALSO**

70452 *grantpt()*, *open()*, *posix_openpt()*, *ptsname()*

70453 XBD <stdlib.h>

70454 **CHANGE HISTORY**

70455 First released in Issue 4, Version 2.

70456 **Issue 5**

70457 Moved from X/OPEN UNIX extension to BASE.

70458 **Issue 6**

70459 The normative text is updated to avoid use of the term ``must'' for application requirements.

70460 **Issue 7**

70461 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0697 [96] is applied.

70462 **NAME**

70463 unsetenv — remove an environment variable

70464 **SYNOPSIS**

```
70465 CX       #include <stdlib.h>
70466       int unsetenv(const char *name);
```

70467 **DESCRIPTION**

70468 The *unsetenv()* function shall remove an environment variable from the environment of the
 70469 calling process. The *name* argument points to a string, which is the name of the variable to be
 70470 removed. The named argument shall not contain an '=' character. If the named variable does
 70471 not exist in the current environment, the environment shall be unchanged and the function is
 70472 considered to have completed successfully.

70473 The *unsetenv()* function shall update the list of pointers to which *environ* points.

70474 The *unsetenv()* function need not be thread-safe.

70475 **RETURN VALUE**

70476 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to
 70477 indicate the error, and the environment shall be unchanged.

70478 **ERRORS**

70479 The *unsetenv()* function shall fail if:

70480	[EINVAL]	The <i>name</i> argument points to an empty string, or points to a string containing
70481		an '=' character.

70482 **EXAMPLES**

70483 None.

70484 **APPLICATION USAGE**

70485 None.

70486 **RATIONALE**

70487 Refer to the RATIONALE section in [setenv\(\)](#).

70488 **FUTURE DIRECTIONS**

70489 None.

70490 **SEE ALSO**

70491 [getenv\(\)](#), [setenv\(\)](#)

70492 XBD [<stdlib.h>](#), [<sys/types.h>](#)

70493 **CHANGE HISTORY**

70494 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

70495 **Issue 7**

70496 Austin Group Interpretation 1003.1-2001 #156 is applied.

70497 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0698 [167] and XSH/TC1-2008/0699
 70498 [185] are applied.

70499 **NAME**

70500 uselocale — use locale in current thread

70501 **SYNOPSIS**

```
70502 CX       #include <locale.h>
70503       locale_t uselocale(locale_t newloc);
```

70504 **DESCRIPTION**70505 The *uselocale()* function shall set or query the current locale for the calling thread.70506 The value for the *newloc* argument shall be one of the following:

- 70507 1. A value returned by the *newlocale()* or *duplocale()* functions
- 70508 2. The special locale object descriptor LC_GLOBAL_LOCALE
- 70509 3. **(locale_t)0**

70510 If the *newloc* argument is **(locale_t)0**, the current locale shall not be changed; this value can be
70511 used to query the current locale setting. If the *newloc* argument is LC_GLOBAL_LOCALE, any
70512 thread-local locale for the calling thread shall be uninstalled; the thread shall again use the
70513 global locale as the current locale, and changes to the global locale shall affect the thread.
70514 Otherwise, the locale represented by *newloc* shall be installed as a thread-local locale to be used
70515 as the current locale for the calling thread.

70516 Once the *uselocale()* function has been called to install a thread-local locale, the behavior of every
70517 interface using data from the current locale shall be affected for the calling thread. The current
70518 locale for other threads shall remain unchanged.

70519 **RETURN VALUE**

70520 Upon successful completion, the *uselocale()* function shall return a handle for the thread-local
70521 locale that was in use as the current locale for the calling thread on entry to the function, or
70522 LC_GLOBAL_LOCALE if no thread-local locale was in use. Otherwise, *uselocale()* shall return
70523 **(locale_t)0** and set *errno* to indicate the error.

70524 **ERRORS**70525 The *uselocale()* function may fail if:

70526 [EINVAL] *newloc* is not a valid locale object and is not **(locale_t)0**.

70527 **EXAMPLES**

70528 None.

70529 **APPLICATION USAGE**

70530 Unlike the *setlocale()* function, the *uselocale()* function does not allow replacing some locale
70531 categories only. Applications that need to install a locale which differs only in a few categories
70532 must use *newlocale()* to change a locale object equivalent to the currently used locale and install
70533 it.

70534 **RATIONALE**

70535 None.

70536 **FUTURE DIRECTIONS**

70537 None.

70538 **SEE ALSO**70539 *duplocale(), freelocale(), newlocale(), setlocale()*70540 XBD <**locale.h**>70541 **CHANGE HISTORY**

70542 First released in Issue 7.

70543 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0700 [290] and XSH/TC1-2008/0701
70544 [334] are applied.

70545 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0382 [582] is applied.

70546 **NAME**

70547 utime ‡set file access and modification times

70548 **SYNOPSIS**

```
70549 OB #include <utime.h>
70550 int utime(const char *path, const struct utimbuf *times);
```

70551 **DESCRIPTION**

70552 The *utime()* function shall set the access and modification times of the file named by the *path*
 70553 argument.

70554 If *times* is a null pointer, the access and modification times of the file shall be set to the current
 70555 time. The effective user ID of the process shall match the owner of the file, or the process has
 70556 write permission to the file or has appropriate privileges, to use *utime()* in this manner.

70557 If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the
 70558 access and modification times shall be set to the values contained in the designated structure.
 70559 Only a process with the effective user ID equal to the user ID of the file or a process with
 70560 appropriate privileges may use *utime()* this way.

70561 The **utimbuf** structure is defined in the **<utime.h>** header. The times in the structure **utimbuf**
 70562 are measured in seconds since the Epoch.

70563 Upon successful completion, the *utime()* function shall mark the last file status change
 70564 timestamp for update; see **<sys/stat.h>**.

70565 **RETURN VALUE**

70566 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* shall
 70567 be set to indicate the error, and the file times shall not be affected.

70568 **ERRORS**

70569 The *utime()* function shall fail if:

70570 [EACCES] Search permission is denied by a component of the path prefix; or the *times*
 70571 argument is a null pointer and the effective user ID of the process does not
 70572 match the owner of the file, the process does not have write permission for the
 70573 file, and the process does not have appropriate privileges.

70574 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 70575 argument.

70576 [ENAMETOOLONG]
 70577 The length of a component of a pathname is longer than {NAME_MAX}.

70578 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

70579 [ENOTDIR] A component of the path prefix names an existing file that is neither a
 70580 directory nor a symbolic link to a directory, or the *path* argument contains at
 70581 least one non-*<slash>* character and ends with one or more trailing *<slash>*
 70582 characters and the last pathname component names an existing file that is
 70583 neither a directory nor a symbolic link to a directory.

70584 [EPERM] The *times* argument is not a null pointer and the effective user ID of the calling
 70585 process does not match the owner of the file and the calling process does not
 70586 have appropriate privileges.

- 70587 [EROFS] The file system containing the file is read-only.
- 70588 The *utime()* function may fail if:
- 70589 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
70590 resolution of the *path* argument.
- 70591 [ENAMETOOLONG]
70592 The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a
70593 symbolic link produced an intermediate result with a length that exceeds
70594 {PATH_MAX}.

70595 EXAMPLES

70596 None.

70597 APPLICATION USAGE

70598 Since the **utimbuf** structure only contains **time_t** variables and is not accurate to fractions of a
70599 second, applications should use the *utimensat()* function instead of the obsolescent *utime()*
70600 function.

70601 RATIONALE

70602 The *actime* structure member must be present so that an application may set it, even though an
70603 implementation may ignore it and not change the last data access timestamp on the file. If an
70604 application intends to leave one of the times of a file unchanged while changing the other, it
70605 should use *stat()* or *fstat()* to retrieve the file's *st_atim* and *st_mtim* parameters, set *actime* and
70606 *modtime* in the buffer, and change one of them before making the *utime()* call.

70607 FUTURE DIRECTIONS

70608 The *utime()* function may be removed in a future version.

70609 SEE ALSO

70610 *fstat()*, *fstatat()*, *futimens()*

70611 XBD <sys/stat.h>, <utime.h>

70612 CHANGE HISTORY

70613 First released in Issue 1. Derived from Issue 1 of the SVID.

70614 Issue 6

70615 The following new requirements on POSIX implementations derive from alignment with the
70616 Single UNIX Specification:

70617 The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
70618 required for conforming implementations of previous POSIX specifications, it was not
70619 required for UNIX applications.

70620 The [ELOOP] mandatory error condition is added.

70621 A second [ENAMETOOLONG] is added as an optional error condition.

70622 The following changes were made to align with the IEEE P1003.1a draft standard:

70623 The [ELOOP] optional error condition is added.

70624 The normative text is updated to avoid use of the term “must” for application requirements.

70625 Issue 7

70626 Austin Group Interpretation 1003.1-2001 #143 is applied.

70627 The *utime()* function is marked obsolescent.

70628 Changes are made related to support for finegrained timestamps.

70629

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0702 [324] is applied.

70630 **NAME**

70631 utimensat, utimes ¶set file access and modification times

70632 **SYNOPSIS**

70633 #include <sys/stat.h>

70634 int utimensat(int *fd*, const char **path*, const struct timespec *times*[2],
70635 int *flag*);

70636 XSI #include <sys/time.h>

70637 int utimes(const char **path*, const struct timeval *times*[2]);

70638 **DESCRIPTION**

70639 Refer to *futimens()*.

70640 **NAME**

70641 va_arg, va_copy, va_end, va_start — handle variable argument list

70642 **SYNOPSIS**

70643 #include <stdarg.h>

70644 type va_arg(va_list ap, type);

70645 void va_copy(va_list dest, va_list src);

70646 void va_end(va_list ap);

70647 void va_start(va_list ap, argN);

70648 **DESCRIPTION**70649 Refer to XBD [<stdarg.h>](#)

70650 **NAME**

70651 `vdprintf, vfprintf, fprintf, vsnprintf, vsprintf` `‡`format output of a `stdarg` argument list

70652 **SYNOPSIS**

70653 `#include <stdarg.h>`

70654 `#include <stdio.h>`

```
70655 CX int vdprintf(int fildest, const char *restrict format, va_list ap);
70656 int vfprintf(FILE *restrict stream, const char *restrict format,
70657 va_list ap);
70658 int fprintf(const char *restrict format, va_list ap);
70659 int vsnprintf(char *restrict s, size_t n, const char *restrict format,
70660 va_list ap);
70661 int vsprintf(char *restrict s, const char *restrict format, va_list ap);
```

70662 **DESCRIPTION**

70663 CX The functionality described on this reference page is aligned with the ISO C standard. Any
70664 conflict between the requirements described here and the ISO C standard is unintentional. This
70665 volume of POSIX.1-2017 defers to the ISO C standard.

70666 CX The `vdprintf()`, `vfprintf()`, `fprintf()`, `vsnprintf()`, and `vsprintf()` functions shall be equivalent to the
70667 CX `dprintf()`, `fprintf()`, `printf()`, `sprintf()`, and `sprintf()` functions respectively, except that instead of
70668 being called with a variable number of arguments, they are called with an argument list as
70669 defined by `<stdarg.h>`.

70670 These functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro,
70671 the value of `ap` after the return is unspecified.

70672 **RETURN VALUE**

70673 Refer to [fprintf\(\)](#).

70674 **ERRORS**

70675 Refer to [fprintf\(\)](#).

70676 **EXAMPLES**

70677 None.

70678 **APPLICATION USAGE**

70679 Applications using these functions should call `va_end(ap)` afterwards to clean up.

70680 **RATIONALE**

70681 None.

70682 **FUTURE DIRECTIONS**

70683 None.

70684 **SEE ALSO**

70685 [Section 2.5](#) (on page 495), [fprintf\(\)](#)

70686 XBD `<stdarg.h>`, `<stdio.h>`

70687 **CHANGE HISTORY**

70688 First released in Issue 1. Derived from Issue 1 of the SVID.

70689 **Issue 5**

70690 The `vsnprintf()` function is added.

70691 **Issue 6**

70692 The *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions are updated for alignment with the
70693 ISO/IEC 9899:1999 standard.

70694 **Issue 7**

70695 The *vdprintf()* function is added to complement the *dprintf()* function from The Open Group
70696 Technical Standard, 2006, Extended API Set Part 1.

70697 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0703 [14] is applied.

70698 **NAME**70699 vfprintf, fprintf, vscanf ⌘format input of a `stdarg` argument list70700 **SYNOPSIS**

70701 #include <stdarg.h>

70702 #include <stdio.h>

70703 int vfprintf(FILE *restrict stream, const char *restrict format,
70704 va_list arg);

70705 int vscanf(const char *restrict format, va_list arg);

70706 int vsscanf(const char *restrict s, const char *restrict format,
70707 va_list arg);70708 **DESCRIPTION**70709 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
70710 conflict between the requirements described here and the ISO C standard is unintentional. This
70711 volume of POSIX.1-2017 defers to the ISO C standard.70712 The `vscanf()`, `vfprintf()`, and `vsscanf()` functions shall be equivalent to the `scanf()`, `fscanf()`, and
70713 `sscanf()` functions, respectively, except that instead of being called with a variable number of
70714 arguments, they are called with an argument list as defined in the `<stdarg.h>` header. These
70715 functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro, the
70716 value of `ap` after the return is unspecified.70717 **RETURN VALUE**70718 Refer to `fscanf()`.70719 **ERRORS**70720 Refer to `fscanf()`.70721 **EXAMPLES**

70722 None.

70723 **APPLICATION USAGE**70724 Applications using these functions should call `va_end(ap)` afterwards to clean up.70725 **RATIONALE**

70726 None.

70727 **FUTURE DIRECTIONS**

70728 None.

70729 **SEE ALSO**70730 Section 2.5 (on page 495), `fscanf()`70731 XBD `<stdarg.h>`, `<stdio.h>`70732 **CHANGE HISTORY**

70733 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

70734 **Issue 7**

70735 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0704 [14] is applied.

70736 **NAME**70737 vfwprintf, vswprintf, vwprintf †wide-character formatted output of a `stdarg` argument list70738 **SYNOPSIS**

```
70739 #include <stdarg.h>
70740 #include <stdio.h>
70741 #include <wchar.h>

70742 int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
70743             va_list arg);
70744 int vswprintf(wchar_t *restrict ws, size_t n,
70745             const wchar_t *restrict format, va_list arg);
70746 int vwprintf(const wchar_t *restrict format, va_list arg);
```

70747 **DESCRIPTION**

70748 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
 70749 conflict between the requirements described here and the ISO C standard is unintentional. This
 70750 volume of POSIX.1-2017 defers to the ISO C standard.

70751 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fwprintf()*, *swprintf()*,
 70752 and *wprintf()* respectively, except that instead of being called with a variable number of
 70753 arguments, they are called with an argument list as defined by **<stdarg.h>**.

70754 These functions shall not invoke the *va_end* macro. However, as these functions do invoke the
 70755 *va_arg* macro, the value of *ap* after the return is unspecified.

70756 **RETURN VALUE**70757 Refer to *fwprintf()*.70758 **ERRORS**70759 Refer to *fwprintf()*.70760 **EXAMPLES**

70761 None.

70762 **APPLICATION USAGE**70763 Applications using these functions should call *va_end(ap)* afterwards to clean up.70764 **RATIONALE**

70765 None.

70766 **FUTURE DIRECTIONS**

70767 None.

70768 **SEE ALSO**70769 [Section 2.5](#) (on page 495), *fwprintf()*70770 XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**70771 **CHANGE HISTORY**

70772 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 70773 (E).

70774 **Issue 6**

70775 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the
 70776 ISO/IEC 9899:1999 standard.

70777 **Issue 7**
70778

POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0705 [14] is applied.

70779 **NAME**70780 vfwscanf, vswscanf, wscanf ‡'wide-character formatted input of a `stdarg` argument list70781 **SYNOPSIS**

70782 #include <stdarg.h>

70783 #include <stdio.h>

70784 #include <wchar.h>

70785 int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
70786 va_list arg);70787 int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
70788 va_list arg);

70789 int wscanf(const wchar_t *restrict format, va_list arg);

70790 **DESCRIPTION**70791 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
70792 conflict between the requirements described here and the ISO C standard is unintentional. This
70793 volume of POSIX.1-2017 defers to the ISO C standard.70794 The *vfwscanf()*, *vswscanf()*, and *wscanf()* functions shall be equivalent to the *fwscanf()*,
70795 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a variable
70796 number of arguments, they are called with an argument list as defined in the **<stdarg.h>** header.
70797 These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro,
70798 the value of *ap* after the return is unspecified.70799 **RETURN VALUE**70800 Refer to *fwscanf()*.70801 **ERRORS**70802 Refer to *fwscanf()*.70803 **EXAMPLES**

70804 None.

70805 **APPLICATION USAGE**70806 Applications using these functions should call *va_end(ap)* afterwards to clean up.70807 **RATIONALE**

70808 None.

70809 **FUTURE DIRECTIONS**

70810 None.

70811 **SEE ALSO**70812 [Section 2.5](#) (on page 495), *fwscanf()*70813 XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**70814 **CHANGE HISTORY**

70815 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

70816 **Issue 7**

70817 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0706 [14] is applied.

70818 **NAME**

70819 vprintf ¶format the output of a `stdarg` argument list

70820 **SYNOPSIS**

70821 #include <stdarg.h>

70822 #include <stdio.h>

70823 int vprintf(const char *restrict *format*, va_list *ap*);

70824 **DESCRIPTION**

70825 Refer to *vfprintf()*.

70826 **NAME**

70827 vscanf †format input of a stdarg argument list

70828 **SYNOPSIS**

70829 #include <stdarg.h>

70830 #include <stdio.h>

70831 int vscanf(const char *restrict *format*, va_list *arg*);70832 **DESCRIPTION**70833 Refer to *vfscanf()*.

70834 **NAME**

70835 vsprintf, vsprintf †'format output of a stdarg argument list

70836 **SYNOPSIS**

70837 #include <stdarg.h>

70838 #include <stdio.h>

70839 int vsnprintf(char *restrict *s*, size_t *n*,70840 const char *restrict *format*, va_list *ap*);70841 int vsprintf(char *restrict *s*, const char *restrict *format*,70842 va_list *ap*);70843 **DESCRIPTION**70844 Refer to *fprintf()*.

70845 **NAME**70846 `vsscanf` `†`format input of a `stdarg` argument list70847 **SYNOPSIS**70848 `#include <stdarg.h>`70849 `#include <stdio.h>`70850 `int vsscanf(const char *restrict s, const char *restrict format,`70851 `va_list arg);`70852 **DESCRIPTION**70853 Refer to [vscanf\(\)](#).

70854 **NAME**70855 `vswprintf` ‡'wide-character formatted output of a `stdarg` argument list70856 **SYNOPSIS**70857 `#include <stdarg.h>`70858 `#include <stdio.h>`70859 `#include <wchar.h>`70860 `int vswprintf(wchar_t *restrict ws, size_t n,`70861 `const wchar_t *restrict format, va_list arg);`70862 **DESCRIPTION**70863 Refer to *[vfwprintf\(\)](#)*.

70864 **NAME**70865 vswscanf †'wide-character formatted input of a `stdarg` argument list70866 **SYNOPSIS**

70867 #include <stdarg.h>

70868 #include <stdio.h>

70869 #include <wchar.h>

70870 int vswscanf(const wchar_t *restrict *ws*, const wchar_t *restrict *format*,
70871 va_list *arg*);70872 **DESCRIPTION**70873 Refer to [vfwscanf\(\)](#).

70874 **NAME**

70875 vwprintf †'wide-character formatted output of a `stdarg` argument list

70876 **SYNOPSIS**

70877 #include <stdarg.h>

70878 #include <stdio.h>

70879 #include <wchar.h>

70880 int vwprintf(const wchar_t *restrict *format*, va_list *arg*);

70881 **DESCRIPTION**

70882 Refer to *vwprintf()*.

70883 **NAME**70884 `vwscanf` `‡`wide-character formatted input of a `stdarg` argument list70885 **SYNOPSIS**70886 `#include <stdarg.h>`70887 `#include <stdio.h>`70888 `#include <wchar.h>`70889 `int vwscanf(const wchar_t *restrict format, va_list arg);`70890 **DESCRIPTION**70891 Refer to [vwscanf\(\)](#).

70892 **NAME**

70893 wait, waitpid — wait for a child process to stop or terminate

70894 **SYNOPSIS**

70895 #include <sys/wait.h>

70896 pid_t wait(int *stat_loc);

70897 pid_t waitpid(pid_t pid, int *stat_loc, int options);

70898 **DESCRIPTION**

70899 The *wait()* and *waitpid()* functions shall obtain status information (see [Section 2.13](#), on page 548) pertaining to one of the caller's child processes. The *wait()* function obtains status information for process termination from any child process. The *waitpid()* function obtains status information for process termination, and optionally process stop and/or continue, from a specified subset of the child processes.

70904 The *wait()* function shall cause the calling thread to become blocked until status information generated by child process termination is made available to the thread, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process, or an error occurs. If termination status information is available prior to the call to *wait()*, return shall be immediate. If termination status information is available for two or more child processes, the order in which their status is reported is unspecified.

70910 As described in [Section 2.13](#) (on page 548), the *wait()* and *waitpid()* functions consume the status information they obtain.

70912 The behavior when multiple threads are blocked in *wait()*, *waitid()*, or *waitpid()* is described in [Section 2.13](#) (on page 548).

70914 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is **(pid_t)-1** and the *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and *options* arguments.

70917 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()* function shall only return the status of a child process from this set:

70919 If *pid* is equal to **(pid_t)-1**, *status* is requested for any child process. In this respect, *waitpid()* is then equivalent to *wait()*.

70921 If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is requested.

70923 If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of the calling process.

70925 If *pid* is less than **(pid_t)-1**, *status* is requested for any child process whose process group ID is equal to the absolute value of *pid*.

70927 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the **<sys/wait.h>** header:

70929 XSI **WCONTINUED** The *waitpid()* function shall report the status of any continued child process specified by *pid* whose status has not been reported since it continued from a job control stop.

70932 **WNOHANG** The *waitpid()* function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*.

70935 WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose
70936 status has not yet been reported since they stopped, shall also be reported to
70937 the requesting process.

70938 If *wait()* or *waitpid()* return because the status of a child process is available, these functions
70939 shall return a value equal to the process ID of the child process. In this case, if the value of the
70940 argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by
70941 *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status
70942 returned is from a terminated child process that terminated by one of the following means:

- 70943 1. The process returned 0 from *main()*.
- 70944 2. The process called *_exit()* or *exit()* with a *status* argument of 0.
- 70945 3. The process was terminated because the last thread in the process terminated.

70946 Regardless of its value, this information may be interpreted using the following macros, which
70947 are defined in `<sys/wait.h>` and evaluate to integral expressions; the *stat_val* argument is the
70948 integer value pointed to by *stat_loc*.

70949 WIFEXITED(*stat_val*)
70950 Evaluates to a non-zero value if *status* was returned for a child process that terminated
70951 normally.

70952 WEXITSTATUS(*stat_val*)
70953 If the value of WIFEXITED(*stat_val*) is non-zero, this macro evaluates to the low-order 8 bits
70954 of the *status* argument that the child process passed to *_exit()* or *exit()*, or the value the child
70955 process returned from *main()*.

70956 WIFSIGNALED(*stat_val*)
70957 Evaluates to a non-zero value if *status* was returned for a child process that terminated due
70958 to the receipt of a signal that was not caught (see `<signal.h>`).

70959 WTERMSIG(*stat_val*)
70960 If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro evaluates to the number of
70961 the signal that caused the termination of the child process.

70962 WIFSTOPPED(*stat_val*)
70963 Evaluates to a non-zero value if *status* was returned for a child process that is currently
70964 stopped.

70965 WSTOPSIG(*stat_val*)
70966 If the value of WIFSTOPPED(*stat_val*) is non-zero, this macro evaluates to the number of the
70967 signal that caused the child process to stop.

70968 XSI WIFCONTINUED(*stat_val*)
70969 Evaluates to a non-zero value if *status* was returned for a child process that has continued
70970 from a job control stop.

70971 SPN It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes
70972 created by *posix_spawn()* or *posix_spawnnp()* can indicate a WIFSTOPPED(*stat_val*) before
70973 subsequent calls to *wait()* or *waitpid()* indicate WIFEXITED(*stat_val*) as the result of an error
70974 detected before the new process image starts executing.

70975 It is unspecified whether the *status* value returned by calls to *wait()* or *waitpid()* for processes
70976 created by *posix_spawn()* or *posix_spawnnp()* can indicate a WIFSIGNALED(*stat_val*) if a signal is
70977 sent to the parent's process group after *posix_spawn()* or *posix_spawnnp()* is called.

70978 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that specified the

70979 XSI WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros
 70980 WIFEXITED(*stat_loc), WIFSIGNALED(*stat_loc), and WIFSTOPPED(*stat_loc) shall evaluate to a
 70981 non-zero value.

70982 XSI If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that specified the
 70983 WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(*stat_loc),
 70984 WIFSIGNALED(*stat_loc), WIFSTOPPED(*stat_loc), and WIFCONTINUED(*stat_loc) shall
 70985 evaluate to a non-zero value.

70986 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 70987 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the
 70988 macros WIFEXITED(*stat_loc) and WIFSIGNALED(*stat_loc) shall evaluate to a non-zero value.

70989 XSI If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 70990 WUNTRACED flag and specified the WCONTINUED flag, exactly one of the macros
 70991 WIFEXITED(*stat_loc), WIFSIGNALED(*stat_loc), and WIFCONTINUED(*stat_loc) shall evaluate
 70992 to a non-zero value.

70993 If `_POSIX_REALTIME_SIGNALS` is defined, and the implementation queues the SIGCHLD
 70994 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any
 70995 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.
 70996 Any other pending SIGCHLD signals shall remain pending.

70997 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child
 70998 process is available, any pending SIGCHLD signal shall be cleared unless the status of another
 70999 child process is available.

71000 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD
 71001 signal is delivered.

71002 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*
 71003 report *status*. This shall not occur unless the calling process or one of its child processes
 71004 explicitly makes use of a non-standard extension. In these cases the interpretation of the
 71005 reported *status* is implementation-defined.

71006 If a parent process terminates without waiting for all of its child processes to terminate, the
 71007 remaining child processes shall be assigned a new parent process ID corresponding to an
 71008 implementation-defined system process.

71009 RETURN VALUE

71010 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions
 71011 shall return a value equal to the process ID of the child process for which *status* is reported. If
 71012 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, `-1` shall be
 71013 returned and *errno* set to `[EINTR]`. If *waitpid()* was invoked with `WNOHANG` set in *options*, it
 71014 has at least one child process specified by *pid* for which *status* is not available, and *status* is not
 71015 available for any process specified by *pid*, `0` is returned. Otherwise, `-1` shall be returned, and
 71016 *errno* set to indicate the error.

71017 ERRORS

71018 The *wait()* function shall fail if:

71019 `[ECHILD]` The calling process has no existing unwaited-for child processes.

71020 `[EINTR]` The function was interrupted by a signal. The value of the location pointed to
 71021 by *stat_loc* is undefined.

71022 The *waitpid()* function shall fail if:

71023	[ECHILD]	The process specified by <i>pid</i> does not exist or is not a child of the calling process, or the process group specified by <i>pid</i> does not exist or does not have any member process that is a child of the calling process.
71024		
71025		
71026	[EINTR]	The function was interrupted by a signal. The value of the location pointed to by <i>stat_loc</i> is undefined.
71027		
71028	[EINVAL]	The <i>options</i> argument is not valid.

71029 **EXAMPLES**71030 **Waiting for a Child Process and then Checking its Status**

71031 The following example demonstrates the use of *waitpid()*, *fork()*, and the macros used to
 71032 interpret the status value returned by *waitpid()* (and *wait()*). The code segment creates a child
 71033 process which does some unspecified work. Meanwhile the parent loops performing calls to
 71034 *waitpid()* to monitor the status of the child. The loop terminates when child termination is
 71035 detected.

```

71036 #include <stdio.h>
71037 #include <stdlib.h>
71038 #include <unistd.h>
71039 #include <sys/wait.h>
71040 ...
71041 pid_t child_pid, wpid;
71042 int status;
71043 child_pid = fork();
71044 if (child_pid == -1) {          /* fork() failed */
71045     perror("fork");
71046     exit(EXIT_FAILURE);
71047 }
71048 if (child_pid == 0) {          /* This is the child */
71049     /* Child does some work and then terminates */
71050     ...
71051 } else {                        /* This is the parent */
71052     do {
71053         wpid = waitpid(child_pid, &status, WUNTRACED
71054 #ifdef WCONTINUED           /* Not all implementations support this */
71055     | WCONTINUED
71056 #endif
71057         );
71058         if (wpid == -1) {
71059             perror("waitpid");
71060             exit(EXIT_FAILURE);
71061         }
71062         if (WIFEXITED(status)) {
71063             printf("child exited, status=%d\n", WEXITSTATUS(status));
71064         } else if (WIFSIGNALED(status)) {
71065             printf("child killed (signal %d)\n", WTERMSIG(status));
71066         } else if (WIFSTOPPED(status)) {
71067             printf("child stopped (signal %d)\n", WSTOPSIG(status));

```

```

71068     #ifndef WIFCONTINUED      /* Not all implementations support this */
71069         } else if (WIFCONTINUED(status)) {
71070             printf("child continued\n");
71071     #endif
71072         } else {      /* Non-standard case -- may never happen */
71073             printf("Unexpected status (0x%x)\n", status);
71074         }
71075     } while (!WIFEXITED(status) && !WIFSIGNALED(status));
71076 }

```

71077 **Waiting for a Child Process in a Signal Handler for SIGCHLD**

71078 The following example demonstrates how to use *waitpid()* in a signal handler for SIGCHLD
71079 without passing *-1* as the *pid* argument. (See the APPLICATION USAGE section below for the
71080 reasons why passing a *pid* of *-1* is not recommended.) The method used here relies on the
71081 standard behavior of *waitpid()* when SIGCHLD is blocked. On historical non-conforming
71082 systems, the status of some child processes might not be reported.

```

71083 #include <stdlib.h>
71084 #include <stdio.h>
71085 #include <signal.h>
71086 #include <sys/types.h>
71087 #include <sys/wait.h>
71088 #include <unistd.h>
71089 #define CHILDREN 10
71090 static void
71091 handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
71092 {
71093     int sav_errno = errno;
71094     int status;
71095     /*
71096      * Obtain status information for the child which
71097      * caused the SIGCHLD signal and write its exit code
71098      * to stdout.
71099      */
71100     if (sinfo->si_code != CLD_EXITED)
71101     {
71102         static char msg[] = "wrong si_code\n";
71103         write(2, msg, sizeof msg - 1);
71104     }
71105     else if (waitpid(sinfo->si_pid, &status, 0) == -1)
71106     {
71107         static char msg[] = "waitpid() failed\n";
71108         write(2, msg, sizeof msg - 1);
71109     }
71110     else if (!WIFEXITED(status))
71111     {
71112         static char msg[] = "WIFEXITED was false\n";
71113         write(2, msg, sizeof msg - 1);
71114     }
71115     else

```

```

71116         {
71117             int code = WEXITSTATUS(status);
71118             char buf[2];
71119             buf[0] = '0' + code;
71120             buf[1] = '\n';
71121             write(1, buf, 2);
71122         }
71123         errno = sav_errno;
71124     }

71125     int
71126     main(void)
71127     {
71128         int i;
71129         pid_t pid;
71130         struct sigaction sa;

71131         sa.sa_flags = SA_SIGINFO;
71132         sa.sa_sigaction = handle_sigchld;
71133         sigemptyset(&sa.sa_mask);
71134         if (sigaction(SIGCHLD, &sa, NULL) == -1)
71135         {
71136             perror("sigaction");
71137             exit(EXIT_FAILURE);
71138         }

71139         for (i = 0; i < CHILDREN; i++)
71140         {
71141             switch (pid = fork())
71142             {
71143                 case -1:
71144                     perror("fork");
71145                     exit(EXIT_FAILURE);
71146                 case 0:
71147                     sleep(2);
71148                     _exit(i);
71149             }
71150         }

71151         /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
71152         alarm(3);
71153         for (;;)
71154             pause();

71155         return 0; /* NOTREACHED */
71156     }

```

71157 APPLICATION USAGE

71158 Calls to *wait()* will collect information about any child process. This may result in interactions
71159 with other interfaces that may be waiting for their own children (such as by use of *system()*). For
71160 this and other reasons it is recommended that portable applications not use *wait()*, but instead
71161 use *waitpid()*. For these same reasons, the use of *waitpid()* with a *pid* argument of *-1*, and the use
71162 of *waitid()* with the *idtype* argument set to *P_ALL*, are also not recommended for portable
71163 applications.

71164 XSI As specified in [Consequences of Process Termination](#) (on page 553), if the calling process has
 71165 SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, then the termination of a child process
 71166 will not cause status information to become available to a thread blocked in *wait()*, *waitid()*, or
 71167 *waitpid()*. Thus, a thread blocked in one of the wait functions will remain blocked unless some
 71168 other condition causes the thread to resume execution (such as an [ECHILD] failure due to no
 71169 remaining children in the set of waited-for children).

71170 RATIONALE

71171 A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the
 71172 calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an
 71173 *exec* or other function calls) from the parent. If a child produces grandchildren by further use of
 71174 *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()*
 71175 from the original parent process. Nothing in this volume of POSIX.1-2017 prevents an
 71176 implementation from providing extensions that permit a process to get *status* from a grandchild
 71177 or any other process, but a process that does not use such extensions must be guaranteed to see
 71178 *status* from only its direct children.

71179 The *waitpid()* function is provided for three reasons:

- 71180 1. To support job control
- 71181 2. To permit a non-blocking version of the *wait()* function
- 71182 3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without
 71183 interfering with other terminated children for which the process has not waited

71184 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The
 71185 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The
 71186 WUNTRACED flag is used only in conjunction with job control on systems supporting job
 71187 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped
 71188 processes in that implementation: processes being traced via the *ptrace()* debugging facility and
 71189 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of
 71190 POSIX.1-2017, only the second type is relevant. The name WUNTRACED was retained because
 71191 its usage is the same, even though the name is not intuitively meaningful in this context.

71192 The third reason for the *waitpid()* function is to permit independent sections of a process to
 71193 spawn and wait for children without interfering with each other. For example, the following
 71194 problem occurs in developing a portable shell, or command interpreter:

```
71195 stream = popen("/bin/true");
71196 (void) system("sleep 100");
71197 (void) pclose(stream);
```

71198 On all historical implementations, the final *pclose()* fails to reap the *wait()* *status* of the *popen()*.

71199 The status values are retrieved by macros, rather than given as specific bit encodings as they are
 71200 in most historical implementations (and thus expected by existing programs). This was
 71201 necessary to eliminate a limitation on the number of signals an implementation can support that
 71202 was inherent in the traditional encodings. This volume of POSIX.1-2017 does require that a *status*
 71203 value of zero corresponds to a process calling *_exit(0)*, as this is the most common encoding
 71204 expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

71205 These macros syntactically operate on an arbitrary integer value. The behavior is undefined
 71206 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed to
 71207 by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each
 71208 argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other
 71209 specifications in this volume of POSIX.1-2017 or traditional usage. It also could have implied

71210 that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or
71211 passed as an argument to other functions before being interpreted by these macros.

71212 The extension that affects *wait()* and *waitpid()* and is common in historical implementations is
71213 the *ptrace()* function. It is called by a child process and causes that child to stop and return a
71214 *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()*
71215 children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()*
71216 function). Most applications do not need to concern themselves with such extensions because
71217 they have control over what extensions they or their children use. However, applications, such
71218 as command interpreters, that invoke arbitrary processes may see this behavior when those
71219 arbitrary processes misuse such extensions.

71220 Implementations that support **core** file creation or other implementation-defined actions on
71221 termination of some processes traditionally provide a bit in the *status* returned by *wait()* to
71222 indicate that such actions have occurred.

71223 Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated
71224 with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()*
71225 category with respect to SIGCHLD.

71226 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the
71227 process in *wait()*, with the same meaning of “accepted” as when that word is applied to the
71228 *sigwait()* family of functions.

71229 Allowing the *wait()* family of functions to behave this way permits an implementation to be able
71230 to deal precisely with SIGCHLD signals.

71231 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the
71232 following guarantees regardless of the queuing depth of signals in general (the list of waitable
71233 children can hold the SIGCHLD queue):

- 71234 1. If a SIGCHLD signal handler is established via *sigaction()* without the SA_RESETHAND
71235 flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal
71236 will be delivered to or accepted by the process for every child process that terminates.
- 71237 2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return
71238 immediately with status information for a child process.
- 71239 3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to
71240 receive a non-null pointer to a **siginfo_t** structure that describes a child process for which
71241 a wait via *waitpid()* or *waitid()* will not block or fail.
- 71242 4. The *system()* function will not cause the SIGCHLD handler of a process to be called as a
71243 result of the *fork()/exec* executed within *system()* because *system()* will accept the
71244 SIGCHLD signal when it performs a *waitpid()* for its child process. This is a desirable
71245 behavior of *system()* so that it can be used in a library without causing side-effects to the
71246 application linked with the library.

71247 An implementation that does not permit the *wait()* family of functions to accept (discard) a
71248 pending SIGCHLD signal associated with a successfully waited-for child, cannot make the
71249 guarantees described above for the following reasons:

71250 Guarantee #1

71251 Although it might be assumed that reliable queuing of all SIGCHLD signals generated by
71252 the system can make this guarantee, the counter-example is the case of a process that blocks
71253 SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the
71254 implementation supports queued signals, then eventually the system will run out of
71255 memory for the queue. The guarantee cannot be made because there must be some limit to

- 71256 the depth of queuing.
- 71257 **Guarantees #2 and #3**
- 71258 These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD
- 71259 signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()*
- 71260 function) will result in an invocation of the handler when SIGCHLD is unblocked, after the
- 71261 process has disappeared.
- 71262 **Guarantee #4**
- 71263 Although possible to make this guarantee, *system()* would have to set the SIGCHLD
- 71264 handler to SIG_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded
- 71265 (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This
- 71266 would have the undesirable side-effect of discarding all SIGCHLD signals pending to the
- 71267 process.
- 71268 **FUTURE DIRECTIONS**
- 71269 None.
- 71270 **SEE ALSO**
- 71271 [Section 2.13](#) (on page 548), *exec*, *exit()*, *fork()*, *system()*, *waitid()*
- 71272 [XBD Section 4.12](#) (on page 111), [<signal.h>](#), [<sys/wait.h>](#)
- 71273 **CHANGE HISTORY**
- 71274 First released in Issue 1. Derived from Issue 1 of the SVID.
- 71275 **Issue 5**
- 71276 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
- 71277 **Issue 6**
- 71278 The following new requirements on POSIX implementations derive from alignment with the
- 71279 Single UNIX Specification:
- 71280 The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
- 71281 required for conforming implementations of previous POSIX specifications, it was not
- 71282 required for UNIX applications.
- 71283 The following changes were made to align with the IEEE P1003.1a draft standard:
- 71284 The processing of the SIGCHLD signal and the [ECHILD] error is clarified.
- 71285 The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*)
- 71286 are defined with respect to *posix_spawn()* or *posix_spawnnp()* for alignment with IEEE Std
- 71287 1003.1d-1999.
- 71288 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.
- 71289 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the
- 71290 EXAMPLES section.
- 71291 **Issue 7**
- 71292 SD5-XSH-ERN-202 is applied.
- 71293 APPLICATION USAGE is added, recommending that the *wait()* function not be used.
- 71294 An additional example for *waitpid()* is added.
- 71295 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0707 [421], XSH/TC1-2008/0708 [166],
- 71296 XSH/TC1-2008/0709 [166], and XSH/TC1-2008/0710 [69] are applied.

71297
71298

POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0384 [690], XSH/TC2-2008/0385 [691], and XSH/TC2-2008/0386 [690] are applied.

71299 **NAME**

71300 waitid — wait for a child process to change state

71301 **SYNOPSIS**

71302 #include <sys/wait.h>

71303 int waitid(idtype_t *idtype*, id_t *id*, siginfo_t **infop*, int *options*);71304 **DESCRIPTION**71305 The *waitid()* function shall obtain status information (see [Section 2.13](#), on page 548) pertaining to
71306 termination, stop, and/or continue events in one of the caller's child processes.71307 The *waitid()* function shall cause the calling thread to become blocked until an error occurs or
71308 status information becomes available to the calling thread that satisfies all of the following
71309 properties ("matching status information"):71310 The status information is from one of the child processes in the set of child processes
71311 specified by the *idtype* and *id* arguments.71312 The state change in the status information matches one of the state change flags set in the
71313 *options* argument.71314 If matching status information is available prior to the call to *waitid()*, return shall be immediate.
71315 If matching status information is available for two or more child processes, the order in which
71316 their status is reported is unspecified.71317 As described in [Section 2.13](#) (on page 548), the *waitid()* function consumes the status information
71318 it obtains unless the WNOWAIT flag is set in the *options* argument.71319 The behavior when multiple threads are blocked in *wait()*, *waitid()*, or *waitpid()* is described in
71320 [Section 2.13](#) (on page 548).71321 The *waitid()* function shall record the obtained status information in the structure pointed to by
71322 *infop*. The fields of the structure pointed to by *infop* shall be filled in as described under "Pointer
71323 to a Function" in [Section 2.4.3](#) (on page 490).71324 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.71325 If *idtype* is P_PID, *waitid()* shall wait for the child with a process ID equal to (**pid_t**)*id*.71326 If *idtype* is P_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid_t**)*id*.71327 If *idtype* is P_ALL, *waitid()* shall wait for any children and *id* is ignored.71328 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed
71329 by OR'ing together the following flags:71330 WCONTINUED Status shall be returned for any continued child process whose status either
71331 has not been reported since it continued from a job control stop or has been
71332 reported only by calls to *waitid()* with the WNOWAIT flag set.

71333 WEXITED Wait for processes that have exited.

71334 WNOHANG Do not hang if no status is available; return immediately.

71335 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This
71336 shall not affect the state of the process; the process may be waited for again
71337 after this call completes.71338 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal,
71339 and whose status either has not been reported since it stopped or has been
71340 reported only by calls to *waitid()* with the WNOWAIT flag set.

71341 Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to
71342 be OR'ed in with the *options* argument.

71343 The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid()*
71344 returns because a child process was found that satisfied the conditions indicated by the
71345 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the
71346 system with the status of the process; the *si_signo* member shall be set equal to SIGCHLD. If
71347 *waitid()* returns because WNOHANG was specified and status is not available for any process
71348 specified by *idtype* and *id*, then the *si_signo* and *si_pid* members of the structure pointed to by
71349 *infop* shall be set to zero and the values of other members of the structure are unspecified.

71350 RETURN VALUE

71351 If WNOHANG was specified and status is not available for any process specified by *idtype* and
71352 *id*, 0 shall be returned. If *waitid()* returns due to the change of state of one of its children, 0 shall
71353 be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

71354 ERRORS

71355 The *waitid()* function shall fail if:

71356	[ECHILD]	The calling process has no existing unwaited-for child processes.
71357	[EINTR]	The <i>waitid()</i> function was interrupted by a signal.
71358	[EINVAL]	An invalid value was specified for <i>options</i> , or <i>idtype</i> and <i>id</i> specify an invalid
71359		set of processes.

71360 EXAMPLES

71361 None.

71362 APPLICATION USAGE

71363 Calls to *waitid()* with *idtype* equal to P_ALL will collect information about any child process.
71364 This may result in interactions with other interfaces that may be waiting for their own children
71365 (such as by use of *system()*). For this reason it is recommended that portable applications not
71366 use *waitid()* with *idtype* of P_ALL. See also APPLICATION USAGE for *wait()*.

71367 XSI As specified in [Consequences of Process Termination](#) (on page 553), if the calling process has
71368 SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, then the termination of a child process
71369 will not cause status information to become available to a thread blocked in *wait()*, *waitid()*, or
71370 *waitpid()*. Thus, a thread blocked in one of the wait functions will remain blocked unless some
71371 other condition causes the thread to resume execution (such as an [ECHILD] failure due to no
71372 remaining children in the set of waited-for children).

71373 RATIONALE

71374 None.

71375 FUTURE DIRECTIONS

71376 None.

71377 SEE ALSO

71378 [Section 2.4.3](#) (on page 490), [Section 2.13](#) (on page 548), *exec*, *exit()*, *wait()*

71379 XBD [<signal.h>](#), [<sys/wait.h>](#)

71380 CHANGE HISTORY

71381 First released in Issue 4, Version 2.

71382 **Issue 5**

71383 Moved from X/OPEN UNIX extension to BASE.

71384 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

71385 **Issue 6**

71386 The normative text is updated to avoid use of the term “must” for application requirements.

71387 **Issue 7**

71388 Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.

71389 The *waitid()* function is moved from the XSI option to the Base.

71390 APPLICATION USAGE is added, recommending that the *waitid()* function not be used with *idtype* equal to P_ALL.

71392 The description of the WNOHANG flag is updated.

71393 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0711 [154], XSH/TC1-2008/0712 [154], and XSH/TC1-2008/0713 [153] are applied.

71395 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0387 [690] is applied.

71396 **NAME**

71397 waitpid — wait for a child process to stop or terminate

71398 **SYNOPSIS**

71399 #include <sys/wait.h>

71400 pid_t waitpid(pid_t *pid*, int **stat_loc*, int *options*);71401 **DESCRIPTION**71402 Refer to *wait()*.

71403 **NAME**

71404 wcpcpy — copy a wide-character string, returning a pointer to its end

71405 **SYNOPSIS**

```
71406 CX       #include <wchar.h>  
71407       wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

71408 **DESCRIPTION**

71409 Refer to *wcscopy()*.

71410 **NAME**

71411 wcpncpy — copy a fixed-size wide-character string, returning a pointer to its end

71412 **SYNOPSIS**

```
71413 CX #include <wchar.h>
71414     wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
71415                     size_t n);
```

71416 **DESCRIPTION**71417 Refer to *wcsncpy()*.

71418 **NAME**

71419 wrtomb — convert a wide-character code to a character (restartable)

71420 **SYNOPSIS**

71421 #include <wchar.h>

71422 size_t wrtomb(char *restrict *s*, wchar_t *wc*, mbstate_t *restrict *ps*);71423 **DESCRIPTION**71424 CX The functionality described on this reference page is aligned with the ISO C standard. Any
71425 conflict between the requirements described here and the ISO C standard is unintentional. This
71426 volume of POSIX.1-2017 defers to the ISO C standard.71427 If *s* is a null pointer, the *wrtomb()* function shall be equivalent to the call:71428 *wrtomb(buf, L'\0', ps)*71429 where *buf* is an internal buffer.71430 If *s* is not a null pointer, the *wrtomb()* function shall determine the number of bytes needed to
71431 represent the character that corresponds to the wide character given by *wc* (including any shift
71432 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At
71433 most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,
71434 preceded by any shift sequence needed to restore the initial shift state. The resulting state
71435 described shall be the initial conversion state.71436 If *ps* is a null pointer, the *wrtomb()* function shall use its own internal **mbstate_t** object, which is
71437 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
71438 pointed to by *ps* shall be used to completely describe the current conversion state of the
71439 associated character sequence. The implementation shall behave as if no function defined in this
71440 volume of POSIX.1-2017 calls *wrtomb()*.71441 CX The *wrtomb()* function need not be thread-safe if called with a NULL *ps* argument.71442 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.71443 The *wrtomb()* function shall not change the setting of *errno* if successful.71444 **RETURN VALUE**71445 The *wrtomb()* function shall return the number of bytes stored in the array object (including any
71446 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this
71447 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)-1;
71448 the conversion state shall be undefined.71449 **ERRORS**71450 The *wrtomb()* function shall fail if:

71451 [EILSEQ] An invalid wide-character code is detected.

71452 The *wrtomb()* function may fail if:71453 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

71454 **EXAMPLES**

71455 None.

71456 **APPLICATION USAGE**

71457 None.

71458 **RATIONALE**

71459 None.

71460 **FUTURE DIRECTIONS**

71461 None.

71462 **SEE ALSO**71463 [mbsinit\(\)](#), [wcsrtombs\(\)](#)71464 XBD <[wchar.h](#)>71465 **CHANGE HISTORY**71466 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
71467 (E).71468 **Issue 6**

71469 In the DESCRIPTION, a note on using this function in a threaded application is added.

71470 Extensions beyond the ISO C standard are marked.

71471 The normative text is updated to avoid use of the term “must” for application requirements.

71472 The *wcr tomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.71473 **Issue 7**71474 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcr tomb()* function
71475 need not be thread-safe if called with a NULL *ps* argument.

71476 Austin Group Interpretation 1003.1-2001 #170 is applied.

71477 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0714 [88] and XSH/TC1-2008/0715
71478 [105] are applied.

71479 **NAME**

71480 wscasecmp, wscasecmp_l, wcsncasecmp, wcsncasecmp_l ‡' case-insensitive wide-character
71481 string comparison

71482 **SYNOPSIS**

```
71483 CX #include <wchar.h>
71484 int wscasecmp(const wchar_t *ws1, const wchar_t *ws2);
71485 int wscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
71486 locale_t locale);
71487 int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
71488 int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
71489 size_t n, locale_t locale);
```

71490 **DESCRIPTION**

71491 The *wscasecmp()* and *wcsncasecmp()* functions are the wide-character equivalent of the
71492 *strcasecmp()* and *strncasecmp()* functions, respectively.

71493 The *wscasecmp()* and *wscasecmp_l()* functions shall compare, while ignoring differences in case,
71494 the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

71495 The *wcsncasecmp()* and *wcsncasecmp_l()* functions shall compare, while ignoring differences in
71496 case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the
71497 wide-character string pointed to by *ws2*.

71498 The *wscasecmp()* and *wcsncasecmp()* functions use the current locale to determine the case of the
71499 wide characters.

71500 The *wscasecmp_l()* and *wcsncasecmp_l()* functions use the locale represented by *locale* to
71501 determine the case of the wide characters.

71502 When the *LC_CTYPE* category of the locale being used is from the POSIX locale, these functions
71503 shall behave as if the wide-character strings had been converted to lowercase and then a
71504 comparison of wide-character codes performed. Otherwise, the results are unspecified.

71505 The information for *wscasecmp_l()* and *wcsncasecmp_l()* about the case of the characters comes
71506 from the locale represented by *locale*.

71507 The behavior is undefined if the *locale* argument to *wscasecmp_l()* or *wcsncasecmp_l()* is the
71508 special locale object *LC_GLOBAL_LOCALE* or is not a valid locale object handle.

71509 **RETURN VALUE**

71510 Upon completion, the *wscasecmp()* and *wscasecmp_l()* functions shall return an integer greater
71511 than, equal to, or less than 0 if the wide-character string pointed to by *ws1* is, ignoring case,
71512 greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

71513 Upon completion, the *wcsncasecmp()* and *wcsncasecmp_l()* functions shall return an integer
71514 greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed
71515 to by *ws1* is, ignoring case, greater than, equal to, or less than the possibly null wide-character
71516 terminated string pointed to by *ws2*, respectively.

71517 No return values are reserved to indicate an error.

71518 **ERRORS**

71519 No errors are defined.

71520 **EXAMPLES**

71521 None.

71522 **APPLICATION USAGE**

71523 None.

71524 **RATIONALE**

71525 None.

71526 **FUTURE DIRECTIONS**

71527 None.

71528 **SEE ALSO**71529 [strcasecmp\(\)](#), [wcscmp\(\)](#), [wcsncmp\(\)](#)71530 XBD [<wchar.h>](#)71531 **CHANGE HISTORY**

71532 First released in Issue 7.

71533 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0716 [294], XSH/TC1-2008/0717 [283],
71534 and XSH/TC1-2008/0718 [283] are applied.

71535 **NAME**

71536 wscat ‡'concatenate two wide-character strings

71537 **SYNOPSIS**

71538 #include <wchar.h>

71539 wchar_t *wscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);

71540 **DESCRIPTION**

71541 CX The functionality described on this reference page is aligned with the ISO C standard. Any
71542 conflict between the requirements described here and the ISO C standard is unintentional. This
71543 volume of POSIX.1-2017 defers to the ISO C standard.

71544 The *wscat()* function shall append a copy of the wide-character string pointed to by *ws2*
71545 (including the terminating null wide-character code) to the end of the wide-character string
71546 pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character
71547 code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is
71548 undefined.

71549 **RETURN VALUE**71550 The *wscat()* function shall return *ws1*; no return value is reserved to indicate an error.71551 **ERRORS**

71552 No errors are defined.

71553 **EXAMPLES**

71554 None.

71555 **APPLICATION USAGE**

71556 None.

71557 **RATIONALE**

71558 None.

71559 **FUTURE DIRECTIONS**

71560 None.

71561 **SEE ALSO**71562 [wcsncat\(\)](#)71563 XBD [<wchar.h>](#)71564 **CHANGE HISTORY**

71565 First released in Issue 4. Derived from the MSE working draft.

71566 **Issue 6**71567 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, *s1* is
71568 changed to *ws1*.71569 The *wscat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

71570 **NAME**71571 `wcschr` ‡wide-character string scanning operation71572 **SYNOPSIS**71573 `#include <wchar.h>`71574 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`71575 **DESCRIPTION**

71576 CX The functionality described on this reference page is aligned with the ISO C standard. Any
71577 conflict between the requirements described here and the ISO C standard is unintentional. This
71578 volume of POSIX.1-2017 defers to the ISO C standard.

71579 The `wcschr()` function shall locate the first occurrence of `wc` in the wide-character string pointed
71580 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type
71581 `wchar_t` and a wide-character code corresponding to a valid character in the current locale. The
71582 terminating null wide-character code is considered to be part of the wide-character string.

71583 **RETURN VALUE**

71584 Upon completion, `wcschr()` shall return a pointer to the wide-character code, or a null pointer if
71585 the wide-character code is not found.

71586 **ERRORS**

71587 No errors are defined.

71588 **EXAMPLES**

71589 None.

71590 **APPLICATION USAGE**

71591 None.

71592 **RATIONALE**

71593 None.

71594 **FUTURE DIRECTIONS**

71595 None.

71596 **SEE ALSO**71597 [wcsrchr\(\)](#)71598 XBD [<wchar.h>](#)71599 **CHANGE HISTORY**

71600 First released in Issue 4. Derived from the MSE working draft.

71601 **Issue 6**

71602 The normative text is updated to avoid use of the term “must” for application requirements.

71603 **NAME**71604 `wscmp` ¶compare two wide-character strings71605 **SYNOPSIS**71606 `#include <wchar.h>`71607 `int wscmp(const wchar_t *ws1, const wchar_t *ws2);`71608 **DESCRIPTION**

71609 CX The functionality described on this reference page is aligned with the ISO C standard. Any
71610 conflict between the requirements described here and the ISO C standard is unintentional. This
71611 volume of POSIX.1-2017 defers to the ISO C standard.

71612 The `wscmp()` function shall compare the wide-character string pointed to by `ws1` to the wide-
71613 character string pointed to by `ws2`.

71614 The sign of a non-zero return value shall be determined by the sign of the difference between the
71615 values of the first pair of wide-character codes that differ in the objects being compared.

71616 **RETURN VALUE**

71617 Upon completion, `wscmp()` shall return an integer greater than, equal to, or less than 0, if the
71618 wide-character string pointed to by `ws1` is greater than, equal to, or less than the wide-character
71619 string pointed to by `ws2`, respectively.

71620 **ERRORS**

71621 No errors are defined.

71622 **EXAMPLES**

71623 None.

71624 **APPLICATION USAGE**

71625 None.

71626 **RATIONALE**

71627 None.

71628 **FUTURE DIRECTIONS**

71629 None.

71630 **SEE ALSO**71631 [wscasecmp\(\)](#), [wcsncmp\(\)](#)71632 XBD [<wchar.h>](#)71633 **CHANGE HISTORY**

71634 First released in Issue 4. Derived from the MSE working draft.

71635 **NAME**71636 `wscoll`, `wscoll_l` ‡wide-character string comparison using collating information71637 **SYNOPSIS**71638 `#include <wchar.h>`71639 `int wscoll(const wchar_t *ws1, const wchar_t *ws2);`71640 CX `int wscoll_l(const wchar_t *ws1, const wchar_t *ws2,`
71641 `locale_t locale);`71642 **DESCRIPTION**71643 CX For `wscoll()`: The functionality described on this reference page is aligned with the ISO C
71644 standard. Any conflict between the requirements described here and the ISO C standard is
71645 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.71646 CX The `wscoll()` and `wscoll_l()` functions shall compare the wide-character string pointed to by
71647 `ws1` to the wide-character string pointed to by `ws2`, both interpreted as appropriate to the
71648 CX `LC_COLLATE` category of the current locale, or the locale represented by `locale`, respectively.71649 CX The `wscoll()` and `wscoll_l()` functions shall not change the setting of `errno` if successful.71650 CX An application wishing to check for error situations should set `errno` to 0 before calling `wscoll()`
71651 or `wscoll_l()`. If `errno` is non-zero on return, an error has occurred.71652 CX The behavior is undefined if the `locale` argument to `wscoll_l()` is the special locale object
71653 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.71654 **RETURN VALUE**71655 CX Upon successful completion, `wscoll()` and `wscoll_l()` shall return an integer greater than, equal
71656 to, or less than 0, according to whether the wide-character string pointed to by `ws1` is greater
71657 than, equal to, or less than the wide-character string pointed to by `ws2`, when both are
71658 CX interpreted as appropriate to the current locale, or to the locale represented by `locale`,
71659 CX respectively. On error, `wscoll()` and `wscoll_l()` shall set `errno`, but no return value is reserved
71660 to indicate an error.71661 **ERRORS**

71662 These functions may fail if:

71663 CX [EINVAL] The `ws1` or `ws2` arguments contain wide-character codes outside the domain of
71664 the collating sequence.71665 **EXAMPLES**

71666 None.

71667 **APPLICATION USAGE**71668 The `wcsxfrm()` and `wscmp()` functions should be used for sorting large lists.71669 **RATIONALE**

71670 None.

71671 **FUTURE DIRECTIONS**

71672 None.

71673 **SEE ALSO**71674 [wscmp\(\)](#), [wcsxfrm\(\)](#)71675 XBD [<wchar.h>](#)

71676 **CHANGE HISTORY**

71677 First released in Issue 4. Derived from the MSE working draft.

71678 **Issue 5**

71679 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

71680 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

71681 **Issue 7**

71682 The *wscoll_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

71684 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0719 [302], XSH/TC1-2008/0720 [283],
71685 and XSH/TC1-2008/0721 [283] are applied.

71686 **NAME**71687 `wcpcpy`, `wcscopy` — copy a wide-character string, returning a pointer to its end71688 **SYNOPSIS**71689 `#include <wchar.h>`71690 CX `wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`71691 `wchar_t *wcscopy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`71692 **DESCRIPTION**71693 CX For `wcscopy()`: The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.71696 CX The `wcpcpy()` and `wcscopy()` functions shall copy the wide-character string pointed to by `ws2` (including the terminating null wide-character code) into the array pointed to by `ws1`.71698 The application shall ensure that there is room for at least `wcslen(ws2)+1` wide characters in the `ws1` array, and that the `ws2` and `ws1` arrays do not overlap.

71700 If copying takes place between objects that overlap, the behavior is undefined.

71701 **RETURN VALUE**71702 CX The `wcpcpy()` function shall return a pointer to the terminating null wide-character code copied into the `ws1` buffer.71704 The `wcscopy()` function shall return `ws1`.

71705 No return values are reserved to indicate an error.

71706 **ERRORS**

71707 No errors are defined.

71708 **EXAMPLES**

71709 None.

71710 **APPLICATION USAGE**

71711 None.

71712 **RATIONALE**

71713 None.

71714 **FUTURE DIRECTIONS**

71715 None.

71716 **SEE ALSO**71717 [strcpy\(\)](#), [wcsdup\(\)](#), [wcsncpy\(\)](#)71718 XBD [<wchar.h>](#)71719 **CHANGE HISTORY**

71720 First released in Issue 4. Derived from the MSE working draft.

71721 **Issue 6**71722 The `wcscopy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.71723 **Issue 7**71724 The `wcpcpy()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

71725

71726 **NAME**

71727 wscspn ‡get the length of a complementary wide substring

71728 **SYNOPSIS**

71729 #include <wchar.h>

71730 size_t wscspn(const wchar_t *ws1, const wchar_t *ws2);

71731 **DESCRIPTION**

71732 CX The functionality described on this reference page is aligned with the ISO C standard. Any
71733 conflict between the requirements described here and the ISO C standard is unintentional. This
71734 volume of POSIX.1-2017 defers to the ISO C standard.

71735 The *wscspn()* function shall compute the length (in wide characters) of the maximum initial
71736 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character
71737 codes *not* from the wide-character string pointed to by *ws2*.

71738 **RETURN VALUE**

71739 The *wscspn()* function shall return the length of the initial substring of *ws1*; no return value is
71740 reserved to indicate an error.

71741 **ERRORS**

71742 No errors are defined.

71743 **EXAMPLES**

71744 None.

71745 **APPLICATION USAGE**

71746 None.

71747 **RATIONALE**

71748 None.

71749 **FUTURE DIRECTIONS**

71750 None.

71751 **SEE ALSO**

71752 *wcsspn()*

71753 XBD <[wchar.h](#)>

71754 **CHANGE HISTORY**

71755 First released in Issue 4. Derived from the MSE working draft.

71756 **Issue 5**

71757 The RETURN VALUE section is updated to indicate that *wscspn()* returns the length of *ws1*,
71758 rather than *ws1* itself.

71759 **NAME**71760 `wcsdup` ‡duplicate a wide-character string71761 **SYNOPSIS**

```
71762 CX #include <wchar.h>
71763      wchar_t *wcsdup(const wchar_t *string);
```

71764 **DESCRIPTION**71765 The `wcsdup()` function is the wide-character equivalent of the `strdup()` function.

71766 The `wcsdup()` function shall return a pointer to a new wide-character string, allocated as if by a
71767 call to `malloc()`, which is the duplicate of the wide-character string `string`. The returned pointer
71768 can be passed to `free()`. A null pointer is returned if the new wide-character string cannot be
71769 created.

71770 **RETURN VALUE**

71771 Upon successful completion, the `wcsdup()` function shall return a pointer to the newly allocated
71772 wide-character string. Otherwise, it shall return a null pointer and set `errno` to indicate the error.

71773 **ERRORS**71774 The `wcsdup()` function shall fail if:

71775 [ENOMEM] Memory large enough for the duplicate string could not be allocated.

71776 **EXAMPLES**

71777 None.

71778 **APPLICATION USAGE**

71779 For functions that allocate memory as if by `malloc()`, the application should release such memory
71780 when it is no longer required by a call to `free()`. For `wcsdup()`, this is the return value.

71781 **RATIONALE**

71782 None.

71783 **FUTURE DIRECTIONS**

71784 None.

71785 **SEE ALSO**71786 [free\(\)](#), [strdup\(\)](#), [wcsncpy\(\)](#)71787 XBD [<wchar.h>](#)71788 **CHANGE HISTORY**

71789 First released in Issue 7.

71790 **NAME**

71791 wcsftime †'convert date and time to a wide-character string

71792 **SYNOPSIS**

71793 #include <wchar.h>

71794 size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
71795 const wchar_t *restrict format, const struct tm *restrict timeptr);71796 **DESCRIPTION**71797 CX The functionality described on this reference page is aligned with the ISO C standard. Any
71798 conflict between the requirements described here and the ISO C standard is unintentional. This
71799 volume of POSIX.1-2017 defers to the ISO C standard.71800 The *wcsftime()* function shall be equivalent to the *strftime()* function, except that:71801 The argument *wcs* points to the initial element of an array of wide characters into which
71802 the generated output is to be placed.71803 The argument *maxsize* indicates the maximum number of wide characters to be placed in
71804 the output array.71805 The argument *format* is a wide-character string and the conversion specifications are
71806 replaced by corresponding sequences of wide characters. It is unspecified whether an
71807 encoding error occurs if the format string contains **wchar_t** values that do not correspond
71808 to members of the character set of the current locale.

71809 CX Field widths specify the number of wide characters instead of the number of bytes.

71810 The return value indicates the number of wide characters placed in the output array.

71811 If copying takes place between objects that overlap, the behavior is undefined.

71812 **RETURN VALUE**71813 If the total number of resulting wide-character codes including the terminating null wide-
71814 character code is no more than *maxsize*, *wcsftime()* shall return the number of wide-character
71815 codes placed into the array pointed to by *wcs*, not including the terminating null wide-character
71816 code. Otherwise, zero is returned and the contents of the array are unspecified.71817 **ERRORS**

71818 No errors are defined.

71819 **EXAMPLES**

71820 None.

71821 **APPLICATION USAGE**

71822 None.

71823 **RATIONALE**

71824 None.

71825 **FUTURE DIRECTIONS**

71826 None.

71827 **SEE ALSO**71828 [strftime\(\)](#)71829 XBD [<wchar.h>](#)

71830 **CHANGE HISTORY**

71831 First released in Issue 4.

71832 **Issue 5**

71833 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

71834 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format*
71835 argument is changed from **const char *** to **const wchar_t ***.

71836 **Issue 6**

71837 The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

71838 **Issue 7**

71839 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0388 [73] and XSH/TC2-2008/0389
71840 [740] are applied.

71841 **NAME**

71842 wcslen, wcsnlen †'get length of a fixed-sized wide-character string

71843 **SYNOPSIS**

71844 #include <wchar.h>

71845 size_t wcslen(const wchar_t *ws);

71846 CX size_t wcsnlen(const wchar_t *ws, size_t maxlen);

71847 **DESCRIPTION**71848 CX For *wcslen()*: The functionality described on this reference page is aligned with the ISO C
71849 standard. Any conflict between the requirements described here and the ISO C standard is
71850 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.71851 The *wcslen()* function shall compute the number of wide-character codes in the wide-character
71852 string to which *ws* points, not including the terminating null wide-character code.71853 CX The *wcsnlen()* function shall compute the smaller of the number of wide characters in the array
71854 to which *ws* points, not including any terminating null wide-character code, and the value of
71855 *maxlen*. The *wcsnlen()* function shall never examine more than the first *maxlen* characters of the
71856 wide-character array pointed to by *ws*.71857 **RETURN VALUE**71858 The *wcslen()* function shall return the length of *ws*.71859 CX The *wcsnlen()* function shall return the number of wide characters preceding the first null wide-
71860 character code in the array to which *ws* points, if *ws* contains a null wide-character code within
71861 the first *maxlen* wide characters; otherwise, it shall return *maxlen*.

71862 No return values are reserved to indicate an error.

71863 **ERRORS**

71864 No errors are defined.

71865 **EXAMPLES**

71866 None.

71867 **APPLICATION USAGE**

71868 None.

71869 **RATIONALE**

71870 None.

71871 **FUTURE DIRECTIONS**

71872 None.

71873 **SEE ALSO**71874 [strlen\(\)](#)71875 XBD [<wchar.h>](#)71876 **CHANGE HISTORY**

71877 First released in Issue 4. Derived from the MSE working draft.

71878 **Issue 7**71879 The *wcsnlen()* function is added from The Open Group Technical Standard, 2006, Extended API
71880 Set Part 1.

71881 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0390 [560] is applied.

71882 **NAME**

71883 wcsncasecmp, wcsncasecmp_l ‡'case-insensitive wide-character string comparison

71884 **SYNOPSIS**

```
71885 CX     #include <wchar.h>
71886       int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
71887       int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
71888                        size_t n, locale_t locale);
```

71889 **DESCRIPTION**71890 Refer to *wcscasecmp()*.

71891 **NAME**

71892 wcsncat ‡'concatenate a wide-character string with part of another

71893 **SYNOPSIS**

71894 #include <wchar.h>

71895 wchar_t *wcsncat(wchar_t *restrict *ws1*, const wchar_t *restrict *ws2*,
71896 size_t *n*);71897 **DESCRIPTION**71898 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
71899 conflict between the requirements described here and the ISO C standard is unintentional. This
71900 volume of POSIX.1-2017 defers to the ISO C standard.71901 The *wcsncat()* function shall append not more than *n* wide-character codes (a null wide-
71902 character code and wide-character codes that follow it are not appended) from the array pointed
71903 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character
71904 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null
71905 wide-character code shall always be appended to the result. If copying takes place between
71906 objects that overlap, the behavior is undefined.71907 **RETURN VALUE**71908 The *wcsncat()* function shall return *ws1*; no return value is reserved to indicate an error.71909 **ERRORS**

71910 No errors are defined.

71911 **EXAMPLES**

71912 None.

71913 **APPLICATION USAGE**

71914 None.

71915 **RATIONALE**

71916 None.

71917 **FUTURE DIRECTIONS**

71918 None.

71919 **SEE ALSO**71920 [wscat\(\)](#)71921 XBD [<wchar.h>](#)71922 **CHANGE HISTORY**

71923 First released in Issue 4. Derived from the MSE working draft.

71924 **Issue 6**71925 The *wcsncat()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

71926 **NAME**71927 `wcsncmp` ¶'compare part of two wide-character strings71928 **SYNOPSIS**71929 `#include <wchar.h>`71930 `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);`71931 **DESCRIPTION**71932 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
71933 conflict between the requirements described here and the ISO C standard is unintentional. This
71934 volume of POSIX.1-2017 defers to the ISO C standard.71935 The `wcsncmp()` function shall compare not more than *n* wide-character codes (wide-character
71936 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*
71937 to the array pointed to by *ws2*.71938 The sign of a non-zero return value shall be determined by the sign of the difference between the
71939 values of the first pair of wide-character codes that differ in the objects being compared.71940 **RETURN VALUE**71941 Upon successful completion, `wcsncmp()` shall return an integer greater than, equal to, or less
71942 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less
71943 than the possibly null-terminated array pointed to by *ws2*, respectively.71944 **ERRORS**

71945 No errors are defined.

71946 **EXAMPLES**

71947 None.

71948 **APPLICATION USAGE**

71949 None.

71950 **RATIONALE**

71951 None.

71952 **FUTURE DIRECTIONS**

71953 None.

71954 **SEE ALSO**71955 [*wscasecmp\(\)*](#), [*wscmp\(\)*](#)71956 XBD [*<wchar.h>*](#)71957 **CHANGE HISTORY**

71958 First released in Issue 4. Derived from the MSE working draft.

71959 **NAME**

71960 wcpncpy, wcsncpy — copy a fixed-size wide-character string, returning a pointer to its end

71961 **SYNOPSIS**

71962 #include <wchar.h>

71963 CX wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
71964 size_t n);71965 wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
71966 size_t n);71967 **DESCRIPTION**71968 CX For *wcsncpy()*: The functionality described on this reference page is aligned with the ISO C
71969 standard. Any conflict between the requirements described here and the ISO C standard is
71970 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.71971 CX The *wcpncpy()* and *wcsncpy()* functions shall copy not more than *n* wide-character codes (wide-
71972 character codes that follow a null wide-character code are not copied) from the array pointed to
71973 by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the
71974 behavior is undefined.71975 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character
71976 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,
71977 until *n* wide-character codes in all are written.71978 **RETURN VALUE**71979 CX If any null wide-character codes were written into the destination, the *wcpncpy()* function shall
71980 return the address of the first such null wide-character code. Otherwise, it shall return *&ws1[n]*.71981 The *wcsncpy()* function shall return *ws1*.

71982 No return values are reserved to indicate an error.

71983 **ERRORS**

71984 No errors are defined.

71985 **EXAMPLES**

71986 None.

71987 **APPLICATION USAGE**71988 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to
71989 by *ws2*, the result is not null-terminated.71990 **RATIONALE**

71991 None.

71992 **FUTURE DIRECTIONS**

71993 None.

71994 **SEE ALSO**71995 *strncpy()*, *wcscpy()*

71996 XBD <wchar.h>

71997 **CHANGE HISTORY**

71998 First released in Issue 4. Derived from the MSE working draft.

71999 **Issue 6**

72000 The *wcsncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

72001 **Issue 7**

72002 The *wcpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API
72003 Set Part 1.

72004 **NAME**

72005 wcsnlen ‡get length of a fixed-sized wide-character string

72006 **SYNOPSIS**

```
72007 CX       #include <wchar.h>
72008       size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

72009 **DESCRIPTION**

72010 Refer to *wcslen()*.

72011 **NAME**

72012 wcsnrtoombs ‡convert wide-character string to multi-byte string

72013 **SYNOPSIS**

```
72014 CX       #include <wchar.h>
72015           size_t wcsnrtoombs(char *restrict dst, const wchar_t **restrict src,
72016                           size_t nwc, size_t len, mbstate_t *restrict ps);
```

72017 **DESCRIPTION**72018 Refer to [wcsrtombs\(\)](#).

72019 **NAME**72020 `wcpbrk` ‡scan a wide-character string for a wide-character code72021 **SYNOPSIS**72022 `#include <wchar.h>`72023 `wchar_t *wcpbrk(const wchar_t *ws1, const wchar_t *ws2);`72024 **DESCRIPTION**

72025 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72026 conflict between the requirements described here and the ISO C standard is unintentional. This
72027 volume of POSIX.1-2017 defers to the ISO C standard.

72028 The `wcpbrk()` function shall locate the first occurrence in the wide-character string pointed to by
72029 `ws1` of any wide-character code from the wide-character string pointed to by `ws2`.

72030 **RETURN VALUE**

72031 Upon successful completion, `wcpbrk()` shall return a pointer to the wide-character code or a null
72032 pointer if no wide-character code from `ws2` occurs in `ws1`.

72033 **ERRORS**

72034 No errors are defined.

72035 **EXAMPLES**

72036 None.

72037 **APPLICATION USAGE**

72038 None.

72039 **RATIONALE**

72040 None.

72041 **FUTURE DIRECTIONS**

72042 None.

72043 **SEE ALSO**72044 [wchr\(\)](#), [wchr\(\)](#)72045 XBD [<wchar.h>](#)72046 **CHANGE HISTORY**

72047 First released in Issue 4. Derived from the MSE working draft.

72048 **NAME**

72049 wcsrchr — wide-character string scanning operation

72050 **SYNOPSIS**

72051 #include <wchar.h>

72052 wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

72053 **DESCRIPTION**

72054 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
72055 conflict between the requirements described here and the ISO C standard is unintentional. This
72056 volume of POSIX.1-2017 defers to the ISO C standard.

72057 The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed
72058 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
72059 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
72060 terminating null wide-character code shall be considered to be part of the wide-character string.

72061 **RETURN VALUE**

72062 Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null
72063 pointer if *wc* does not occur in the wide-character string.

72064 **ERRORS**

72065 No errors are defined.

72066 **EXAMPLES**

72067 None.

72068 **APPLICATION USAGE**

72069 None.

72070 **RATIONALE**

72071 None.

72072 **FUTURE DIRECTIONS**

72073 None.

72074 **SEE ALSO**72075 [wcschr\(\)](#)72076 XBD [<wchar.h>](#)72077 **CHANGE HISTORY**

72078 First released in Issue 4. Derived from the MSE working draft.

72079 **Issue 6**

72080 The normative text is updated to avoid use of the term “must” for application requirements.

72081 **NAME**

72082 wcsnrtoombs, wcsrtoombs — convert a wide-character string to a character string (restartable)

72083 **SYNOPSIS**

72084 #include <wchar.h>

```
72085 CX     size_t wcsnrtoombs(char *restrict dst, const wchar_t **restrict src,
72086                         size_t nwc, size_t len, mbstate_t *restrict ps);
72087     size_t wcsrtoombs(char *restrict dst, const wchar_t **restrict src,
72088                         size_t len, mbstate_t *restrict ps);
```

72089 **DESCRIPTION**

72090 CX For *wcsrtoombs()*: The functionality described on this reference page is aligned with the ISO C
72091 standard. Any conflict between the requirements described here and the ISO C standard is
72092 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

72093 The *wcsrtoombs()* function shall convert a sequence of wide characters from the array indirectly
72094 pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state
72095 described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters
72096 shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a
72097 terminating null wide character, which shall also be stored. Conversion shall stop earlier in the
72098 following cases:

72099 When a code is reached that does not correspond to a valid character

72100 When the next character would exceed the limit of *len* total bytes to be stored in the array
72101 pointed to by *dst* (and *dst* is not a null pointer)

72102 Each conversion shall take place as if by a call to the *wcrtomb()* function.

72103 If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null
72104 pointer (if conversion stopped due to reaching a terminating null wide character) or the address
72105 just past the last wide character converted (if any). If conversion stopped due to reaching a
72106 terminating null wide character, the resulting state described shall be the initial conversion state.

72107 If *ps* is a null pointer, the *wcsrtoombs()* function shall use its own internal **mbstate_t** object, which
72108 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
72109 pointed to by *ps* shall be used to completely describe the current conversion state of the
72110 associated character sequence.

72111 CX The *wcsnrtoombs()* and *wcsrtoombs()* functions need not be thread-safe if called with a NULL *ps*
72112 argument.

72113 The *wcsnrtoombs()* function shall be equivalent to the *wcsrtoombs()* function, except that the
72114 conversion is limited to the first *nwc* wide characters.

72115 The *wcsrtoombs()* function shall not change the setting of *errno* if successful.

72116 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

72117 The implementation shall behave as if no function defined in System Interfaces volume of
72118 POSIX.1-2017 calls these functions.

72119 **RETURN VALUE**

72120 If conversion stops because a code is reached that does not correspond to a valid character, an
72121 encoding error occurs. In this case, these functions shall store the value of the macro [EILSEQ] in
72122 *errno* and return (**size_t**)-1; the conversion state is undefined. Otherwise, these functions shall
72123 return the number of bytes in the resulting character sequence, not including the terminating
72124 null (if any).

72125 **ERRORS**

72126 These functions shall fail if:

72127 [EILSEQ] A wide-character code does not correspond to a valid character.

72128 These functions may fail if:

72129 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.72130 **EXAMPLES**

72131 None.

72132 **APPLICATION USAGE**

72133 None.

72134 **RATIONALE**

72135 None.

72136 **FUTURE DIRECTIONS**

72137 None.

72138 **SEE ALSO**72139 *mbsinit()*, *wcrtomb()*72140 XBD <[wchar.h](#)>72141 **CHANGE HISTORY**72142 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
72143 (E).72144 **Issue 6**

72145 In the DESCRIPTION, a note on using this function in a threaded application is added.

72146 Extensions beyond the ISO C standard are marked.

72147 The normative text is updated to avoid use of the term “must” for application requirements.

72148 The *wcsrtombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.72149 **Issue 7**72150 Austin Group Interpretation 1003.1-2001 #148 is applied, clarifying that the *wcsrtombs()* function
72151 need not be thread-safe if called with a NULL *ps* argument.

72152 Austin Group Interpretation 1003.1-2001 #170 is applied.

72153 The *wcsnrombs()* function is added from The Open Group Technical Standard, 2006, Extended
72154 API Set Part 1.

72155 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0722 [109,105] is applied.

72156 **NAME**

72157 wcsspn †'get the length of a wide substring

72158 **SYNOPSIS**

72159 #include <wchar.h>

72160 size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);

72161 **DESCRIPTION**

72162 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72163 conflict between the requirements described here and the ISO C standard is unintentional. This
72164 volume of POSIX.1-2017 defers to the ISO C standard.

72165 The *wcsspn()* function shall compute the length (in wide characters) of the maximum initial
72166 segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character
72167 codes from the wide-character string pointed to by *ws2*.

72168 **RETURN VALUE**

72169 The *wcsspn()* function shall return the length of the initial substring of *ws1*; no return value is
72170 reserved to indicate an error.

72171 **ERRORS**

72172 No errors are defined.

72173 **EXAMPLES**

72174 None.

72175 **APPLICATION USAGE**

72176 None.

72177 **RATIONALE**

72178 None.

72179 **FUTURE DIRECTIONS**

72180 None.

72181 **SEE ALSO**72182 [wcscspn\(\)](#)72183 XBD [<wchar.h>](#)72184 **CHANGE HISTORY**

72185 First released in Issue 4. Derived from the MSE working draft.

72186 **Issue 5**

72187 The RETURN VALUE section is updated to indicate that *wcsspn()* returns the length of *ws1*
72188 rather than *ws1* itself.

72189 **NAME**72190 `wcsstr` ¶find a wide-character substring72191 **SYNOPSIS**72192 `#include <wchar.h>`72193 `wchar_t *wcsstr(const wchar_t *restrict ws1,`
72194 `const wchar_t *restrict ws2);`72195 **DESCRIPTION**72196 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72197 conflict between the requirements described here and the ISO C standard is unintentional. This
72198 volume of POSIX.1-2017 defers to the ISO C standard.72199 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by
72200 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the
72201 wide-character string pointed to by `ws2`.72202 **RETURN VALUE**72203 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,
72204 or a null pointer if the wide-character string is not found.72205 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.72206 **ERRORS**

72207 No errors are defined.

72208 **EXAMPLES**

72209 None.

72210 **APPLICATION USAGE**

72211 None.

72212 **RATIONALE**

72213 None.

72214 **FUTURE DIRECTIONS**

72215 None.

72216 **SEE ALSO**72217 [wcschr\(\)](#)72218 XBD [<wchar.h>](#)72219 **CHANGE HISTORY**72220 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
72221 (E).72222 **Issue 6**72223 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

72224 **NAME**

72225 wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

72226 **SYNOPSIS**

72227 #include <wchar.h>

72228 double wcstod(const wchar_t *restrict *nptr*, wchar_t **restrict *endptr*);72229 float wcstof(const wchar_t *restrict *nptr*, wchar_t **restrict *endptr*);72230 long double wcstold(const wchar_t *restrict *nptr*,72231 wchar_t **restrict *endptr*);72232 **DESCRIPTION**72233 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
72234 conflict between the requirements described here and the ISO C standard is unintentional. This
72235 volume of POSIX.1-2017 defers to the ISO C standard.72236 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
72237 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the
72238 input wide-character string into three parts:

- 72239 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
-
- 72240
- isspace()*
-)
-
- 72241 2. A subject sequence interpreted as a floating-point constant or representing infinity or
-
- 72242 NaN
-
- 72243 3. A final wide-character string of one or more unrecognized wide-character codes,
-
- 72244 including the terminating null wide-character code of the input wide-character string

72245 Then they shall attempt to convert the subject sequence to a floating-point number, and return
72246 the result.72247 The expected form of the subject sequence is an optional '+' or '-' sign, then one of the
72248 following:72249 A non-empty sequence of decimal digits optionally containing a radix character; then an
72250 optional exponent part consisting of the wide character 'e' or the wide character 'E',
72251 optionally followed by a '+' or '-' wide character, and then followed by one or more
72252 decimal digits72253 A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix
72254 character; then an optional binary exponent part consisting of the wide character 'p' or
72255 the wide character 'P', optionally followed by a '+' or '-' wide character, and then
72256 followed by one or more decimal digits

72257 One of INF or INFINITY, or any other wide string equivalent except for case

72258 One of NAN or NAN(*n-wchar-sequence_{opt}*), or any other wide string ignoring case in the
72259 NAN part, where:

72260 n-wchar-sequence:

72261 digit

72262 nondigit

72263 n-wchar-sequence digit

72264 n-wchar-sequence nondigit

72265 The subject sequence is defined as the longest initial subsequence of the input wide string,
72266 starting with the first non-white-space wide character, that is of the expected form. The subject
72267 sequence contains no wide characters if the input wide string is not of the expected form.

72268 If the subject sequence has the expected form for a floating-point number, the sequence of wide
 72269 characters starting with the first digit or the radix character (whichever occurs first) shall be
 72270 interpreted as a floating constant according to the rules of the C language, except that the radix
 72271 character shall be used in place of a period, and that if neither an exponent part nor a radix
 72272 character appears in a decimal floating-point number, or if a binary exponent part does not
 72273 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with
 72274 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins
 72275 with a <hyphen-minus>, the sequence shall be interpreted as negated. A wide-character
 72276 sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type,
 72277 else as if it were a floating constant that is too large for the range of the return type. A wide-
 72278 character sequence NAN or NAN(*n-wchar-sequence_{opt}*) shall be interpreted as a quiet NaN, if
 72279 supported in the return type, else as if it were a subject sequence part that does not have the
 72280 expected form; the meaning of the *n-wchar* sequences is implementation-defined. A pointer to
 72281 the final wide string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not
 72282 a null pointer.

72283 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the
 72284 conversion shall be rounded in an implementation-defined manner.

72285 CX The radix character shall be as defined in the current locale (category *LC_NUMERIC*). In the
 72286 POSIX locale, or in a locale where the radix character is not defined, the radix character shall
 72287 default to a <period> (' . ').

72288 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
 72289 accepted.

72290 If the subject sequence is empty or does not have the expected form, no conversion shall be
 72291 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
 72292 *endptr* is not a null pointer.

72293 These functions shall not change the setting of *errno* if successful.

72294 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 72295 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check
 72296 *errno*.

72297 RETURN VALUE

72298 Upon successful completion, these functions shall return the converted value. If no conversion
 72299 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

72300 If the correct value is outside the range of representable values, \pm HUGE_VAL, \pm HUGE_VALF, or
 72301 \pm HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to
 72302 [ERANGE].

72303 If the correct value would cause underflow, a value whose magnitude is no greater than the
 72304 smallest normalized positive number in the return type shall be returned and *errno* set to
 72305 [ERANGE].

72306 ERRORS

72307 The *wcstod()* function shall fail if:

72308 [ERANGE] The value to be returned would cause overflow or underflow.

72309 The *wcstod()* function may fail if:

72310 CX [EINVAL] No conversion could be performed.

72311 **EXAMPLES**

72312 None.

72313 **APPLICATION USAGE**

72314 If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the
 72315 result is not exactly representable, the result should be one of the two numbers in the
 72316 appropriate internal format that are adjacent to the hexadecimal floating source value, with the
 72317 extra stipulation that the error should have a correct sign for the current rounding direction.

72318 If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in `<float.h>`)
 72319 significant digits, the result should be correctly rounded. If the subject sequence *D* has the
 72320 decimal form and more than DECIMAL_DIG significant digits, consider the two bounding,
 72321 adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the
 72322 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or
 72323 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current
 72324 rounding direction, with the extra stipulation that the error with respect to *D* should have a
 72325 correct sign for the current rounding direction.

72326 **RATIONALE**

72327 None.

72328 **FUTURE DIRECTIONS**

72329 None.

72330 **SEE ALSO**72331 [*fscanf\(\)*](#), [*iswspace\(\)*](#), [*localeconv\(\)*](#), [*setlocale\(\)*](#), [*wcstol\(\)*](#)72332 XBD [Chapter 7](#) (on page 135), `<float.h>`, `<wchar.h>`72333 **CHANGE HISTORY**

72334 First released in Issue 4. Derived from the MSE working draft.

72335 **Issue 5**72336 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.72337 **Issue 6**

72338 Extensions beyond the ISO C standard are marked.

72339 The following new requirements on POSIX implementations derive from alignment with the
 72340 Single UNIX Specification:

72341 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
 72342 added if no conversion could be performed.

72343 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

72344 The *wcstod()* prototype is updated.72345 The *wcstof()* and *wcstold()* functions are added.

72346 If the correct value for *wcstod()* would cause underflow, the return value changed from 0
 72347 (as specified in Issue 5) to the smallest normalized positive number.

72348 The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are
 72349 extensively updated.

72350 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

72351 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second
 72352 paragraph in the RETURN VALUE section.

72353 **Issue 7**

72354 Austin Group Interpretation 1003.1-2001 #015 is applied.

72355 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0723 [302] and XSH/TC1-2008/0724
72356 [105] are applied.

72357 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0391 [584] and XSH/TC2-2008/0392
72358 [796] are applied.

72359 **NAME**

72360 wcstoimax, wcstoumax ‡convert a wide-character string to an integer type

72361 **SYNOPSIS**

```
72362       #include <stddef.h>
72363       #include <inttypes.h>
72364       intmax_t wcstoimax(const wchar_t *restrict nptr,
72365                        wchar_t **restrict endptr, int base);
72366       uintmax_t wcstoumax(const wchar_t *restrict nptr,
72367                        wchar_t **restrict endptr, int base);
```

72368 **DESCRIPTION**

72369 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72370 conflict between the requirements described here and the ISO C standard is unintentional. This
72371 volume of POSIX.1-2017 defers to the ISO C standard.

72372 These functions shall be equivalent to the *wcstol()*, *wcstoll()*, *wcstoul()*, and *wcstoull()* functions,
72373 respectively, except that the initial portion of the wide string shall be converted to **intmax_t** and
72374 **uintmax_t** representation, respectively.

72375 **RETURN VALUE**

72376 These functions shall return the converted value, if any.

72377 If no conversion could be performed, zero shall be returned. If the correct value is outside the
72378 range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall
72379 be returned (according to the return type and sign of the value, if any), and *errno* shall be set to
72380 [ERANGE].

72381 **ERRORS**

72382 These functions shall fail if:

72383 [EINVAL] The value of *base* is not supported.

72384 [ERANGE] The value to be returned is not representable.

72385 These functions may fail if:

72386 [EINVAL] No conversion could be performed.

72387 **EXAMPLES**

72388 None.

72389 **APPLICATION USAGE**

72390 None.

72391 **RATIONALE**

72392 None.

72393 **FUTURE DIRECTIONS**

72394 None.

72395 **SEE ALSO**72396 *wcstol()*, *wcstoul()*

72397 XBD <inttypes.h>, <stddef.h>

72398 **CHANGE HISTORY**

72399 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

72400 **NAME**

72401 wcstok ‡split a wide-character string into tokens

72402 **SYNOPSIS**

72403 #include <wchar.h>

72404 wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
72405 wchar_t **restrict ptr);72406 **DESCRIPTION**72407 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72408 conflict between the requirements described here and the ISO C standard is unintentional. This
72409 volume of POSIX.1-2017 defers to the ISO C standard.72410 A sequence of calls to *wcstok()* shall break the wide-character string pointed to by *ws1* into a
72411 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-
72412 character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar_t** pointer
72413 into which the *wcstok()* function shall store information necessary for it to continue scanning the
72414 same wide-character string.72415 The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null
72416 pointer as their first argument. The separator string pointed to by *ws2* may be different from call
72417 to call.72418 The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first
72419 wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no
72420 such wide-character code is found, then there are no tokens in the wide-character string pointed
72421 to by *ws1* and *wcstok()* shall return a null pointer. If such a wide-character code is found, it shall
72422 be the start of the first token.72423 The *wcstok()* function shall then search from there for a wide-character code that *is* contained in
72424 the current separator string. If no such wide-character code is found, the current token extends
72425 to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token
72426 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a
72427 null wide character, which terminates the current token. The *wcstok()* function shall save a
72428 pointer to the following wide-character code, from which the next search for a token shall start.72429 Each subsequent call, with a null pointer as the value of the first argument, shall start searching
72430 from the saved pointer and behave as described above.72431 The implementation shall behave as if no function calls *wcstok()*.72432 **RETURN VALUE**72433 Upon successful completion, the *wcstok()* function shall return a pointer to the first wide-
72434 character code of a token. Otherwise, if there is no token, *wcstok()* shall return a null pointer.72435 **ERRORS**

72436 No errors are defined.

72437 **EXAMPLES**

72438 None.

72439 **APPLICATION USAGE**

72440 None.

72441 **RATIONALE**

72442 None.

72443 **FUTURE DIRECTIONS**

72444 None.

72445 **SEE ALSO**72446 XBD [<wchar.h>](#)72447 **CHANGE HISTORY**

72448 First released in Issue 4.

72449 **Issue 5**72450 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is
72451 added to the definition of *wcstok()* in the SYNOPSIS.72452 **Issue 6**72453 The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

72454 **NAME**72455 `wcstol, wcstoll` ‡convert a wide-character string to a long integer72456 **SYNOPSIS**72457 `#include <wchar.h>`72458 `long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,`
72459 `int base);`72460 `long long wcstoll(const wchar_t *restrict nptr,`
72461 `wchar_t **restrict endptr, int base);`72462 **DESCRIPTION**72463 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72464 conflict between the requirements described here and the ISO C standard is unintentional. This
72465 volume of POSIX.1-2017 defers to the ISO C standard.72466 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
72467 **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

- 72468 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
-
- 72469
- `iswspace()`
-)
-
- 72470 2. A subject sequence interpreted as an integer represented in some radix determined by the
-
- 72471 value of
- base*
-
- 72472 3. A final wide-character string of one or more unrecognized wide-character codes,
-
- 72473 including the terminating null wide-character code of the input wide-character string

72474 Then they shall attempt to convert the subject sequence to an integer, and return the result.

72475 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
72476 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
72477 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
72478 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
72479 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
72480 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.72481 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
72482 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
72483 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
72484 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
72485 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code
72486 representations of 0x or 0X may optionally precede the sequence of letters and digits, following
72487 the sign if present.72488 The subject sequence is defined as the longest initial subsequence of the input wide-character
72489 string, starting with the first non-white-space wide-character code that is of the expected form.
72490 The subject sequence contains no wide-character codes if the input wide-character string is
72491 empty or consists entirely of white-space wide-character code, or if the first non-white-space
72492 wide-character code is other than a sign or a permissible letter or digit.72493 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
72494 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
72495 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
72496 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
72497 <hyphen-minus>, the value resulting from the conversion shall be negated. A pointer to the final
72498 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is
72499 not a null pointer.

72500 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
72501 accepted.

72502 If the subject sequence is empty or does not have the expected form, no conversion shall be
72503 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
72504 *endptr* is not a null pointer.

72505 These functions shall not change the setting of *errno* if successful.

72506 Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on
72507 error and are also valid returns on success, an application wishing to check for error situations
72508 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.

72509 RETURN VALUE

72510 Upon successful completion, these functions shall return the converted value, if any. If no
72511 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If
72512 the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
72513 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
72514 *errno* set to [ERANGE].

72515 ERRORS

72516 These functions shall fail if:

72517 CX [EINVAL] The value of *base* is not supported.

72518 [ERANGE] The value to be returned is not representable.

72519 These functions may fail if:

72520 CX [EINVAL] No conversion could be performed.

72521 EXAMPLES

72522 None.

72523 APPLICATION USAGE

72524 None.

72525 RATIONALE

72526 None.

72527 FUTURE DIRECTIONS

72528 None.

72529 SEE ALSO

72530 *fscanf()*, *iswalph()*, *wcstod()*

72531 XBD <wchar.h>

72532 CHANGE HISTORY

72533 First released in Issue 4. Derived from the MSE working draft.

72534 Issue 5

72535 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

72536 Issue 6

72537 Extensions beyond the ISO C standard are marked.

72538 The following new requirements on POSIX implementations derive from alignment with the
72539 Single UNIX Specification:

72540 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
72541 added if no conversion could be performed.

72542 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

72543 The *wcstol()* prototype is updated.

72544 The *wcstoll()* function is added.

72545 **Issue 7**

72546 SD5-XSH-ERN-56 is applied, removing the reference to **unsigned long** and **unsigned long long**
72547 from the DESCRIPTION.

72548 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0725 [105] is applied.

72549 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0393 [584] and XSH/TC2-2008/0394
72550 [796] are applied.

72551 **NAME**

72552 wcstold — convert a wide-character string to a double-precision number

72553 **SYNOPSIS**

72554 #include <wchar.h>

72555 long double wcstold(const wchar_t *restrict *nptr*,72556 wchar_t **restrict *endptr*);72557 **DESCRIPTION**72558 Refer to *wcstod()*.

72559 **NAME**

72560 wcstoll ¶convert a wide-character string to a long integer

72561 **SYNOPSIS**

72562 #include <wchar.h>

72563 long long wcstoll(const wchar_t *restrict *nptr*,72564 wchar_t **restrict *endptr*, int *base*);72565 **DESCRIPTION**72566 Refer to *wcstol()*.

72567 **NAME**

72568 wcstombs †convert a wide-character string to a character string

72569 **SYNOPSIS**

72570 #include <stdlib.h>

72571 size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,
72572 size_t n);72573 **DESCRIPTION**72574 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72575 conflict between the requirements described here and the ISO C standard is unintentional. This
72576 volume of POSIX.1-2017 defers to the ISO C standard.72577 The *wcstombs()* function shall convert the sequence of wide-character codes that are in the array
72578 pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store
72579 these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n*
72580 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to
72581 *wctomb()*, except that the shift state of *wctomb()* shall not be affected.72582 The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.72583 No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place
72584 CX between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs()* shall
72585 return the length required to convert the entire array regardless of the value of *n*, but no values
72586 are stored.72587 **RETURN VALUE**72588 If a wide-character code is encountered that does not correspond to a valid character (of one or
72589 more bytes each), *wcstombs()* shall return (**size_t**)-1. Otherwise, *wcstombs()* shall return the
72590 number of bytes stored in the character array, not including any terminating null byte. The array
72591 shall not be null-terminated if the value returned is *n*.72592 **ERRORS**72593 The *wcstombs()* function shall fail if:

72594 CX [EILSEQ] A wide-character code does not correspond to a valid character.

72595 **EXAMPLES**

72596 None.

72597 **APPLICATION USAGE**

72598 None.

72599 **RATIONALE**

72600 None.

72601 **FUTURE DIRECTIONS**

72602 None.

72603 **SEE ALSO**72604 *mblen()*, *mbtowc()*, *mbstowcs()*, *wctomb()*

72605 XBD <stdlib.h>

72606 **CHANGE HISTORY**

72607 First released in Issue 4. Derived from the ISO C standard.

72608 **Issue 6**

72609 The following new requirements on POSIX implementations derive from alignment with the
72610 Single UNIX Specification:

72611 The DESCRIPTION states the effect of when *s* is a null pointer.

72612 The [EILSEQ] error condition is added.

72613 The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

72614 **Issue 7**

72615 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

72616 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0726 [109] is applied.

72617 **NAME**

72618 wcstoul, wcstoull †'convert a wide-character string to an unsigned long

72619 **SYNOPSIS**

72620 #include <wchar.h>

72621 unsigned long wcstoul(const wchar_t *restrict *nptr*,
72622 wchar_t **restrict *endptr*, int *base*);72623 unsigned long long wcstoull(const wchar_t *restrict *nptr*,
72624 wchar_t **restrict *endptr*, int *base*);72625 **DESCRIPTION**72626 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
72627 conflict between the requirements described here and the ISO C standard is unintentional. This
72628 volume of POSIX.1-2017 defers to the ISO C standard.72629 The *wcstoul()* and *wcstoull()* functions shall convert the initial portion of the wide-character
72630 string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively.
72631 First, they shall decompose the input wide-character string into three parts:

- 72632 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
-
- 72633
- iswspace()*
-)
-
- 72634 2. A subject sequence interpreted as an integer represented in some radix determined by the
-
- 72635 value of
- base*
-
- 72636 3. A final wide-character string of one or more unrecognized wide-character codes,
-
- 72637 including the terminating null wide-character code of the input wide-character string

72638 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
72639 result.72640 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
72641 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
72642 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
72643 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
72644 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
72645 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.72646 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
72647 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
72648 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
72649 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
72650 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X
72651 may optionally precede the sequence of letters and digits, following the sign if present.72652 The subject sequence is defined as the longest initial subsequence of the input wide-character
72653 string, starting with the first wide-character code that is not white space and is of the expected
72654 form. The subject sequence contains no wide-character codes if the input wide-character string is
72655 empty or consists entirely of white-space wide-character codes, or if the first wide-character
72656 code that is not white space is other than a sign or a permissible letter or digit.72657 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
72658 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
72659 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
72660 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
72661 <hyphen-minus>, the value resulting from the conversion shall be negated. A pointer to the final
72662 wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is

- 72663 not a null pointer.
- 72664 CX In other than the C or POSIX locale, additional locale-specific subject sequence forms may be
72665 accepted.
- 72666 If the subject sequence is empty or does not have the expected form, no conversion shall be
72667 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
72668 *endptr* is not a null pointer.
- 72669 These functions shall not change the setting of *errno* if successful.
- 72670 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return
72671 on success, an application wishing to check for error situations should set *errno* to 0, then call
72672 *wcstoul()* or *wcstoull()*, then check *errno*.
- 72673 **RETURN VALUE**
- 72674 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted
72675 CX value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to
72676 indicate the error. If the correct value is outside the range of representable values,
72677 {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to [ERANGE].
- 72678 **ERRORS**
- 72679 These functions shall fail if:
- 72680 CX [EINVAL] The value of *base* is not supported.
- 72681 [ERANGE] The value to be returned is not representable.
- 72682 These functions may fail if:
- 72683 CX [EINVAL] No conversion could be performed.
- 72684 **EXAMPLES**
- 72685 None.
- 72686 **APPLICATION USAGE**
- 72687 None.
- 72688 **RATIONALE**
- 72689 None.
- 72690 **FUTURE DIRECTIONS**
- 72691 None.
- 72692 **SEE ALSO**
- 72693 [fscanf\(\)](#), [iswalphalpha\(\)](#), [wcstod\(\)](#), [wcstol\(\)](#)
- 72694 XBD <[wchar.h](#)>
- 72695 **CHANGE HISTORY**
- 72696 First released in Issue 4. Derived from the MSE working draft.
- 72697 **Issue 5**
- 72698 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
- 72699 **Issue 6**
- 72700 Extensions beyond the ISO C standard are marked.

72701 The following new requirements on POSIX implementations derive from alignment with the
72702 Single UNIX Specification:

72703 The [EINVAL] error condition is added for when the value of *base* is not supported.

72704 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
72705 added if no conversion could be performed.

72706 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

72707 The *wcstoul()* prototype is updated.

72708 The *wcstoull()* function is added.

72709 **Issue 7**

72710 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0727 [105] is applied.

72711 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0395 [584] and XSH/TC2-2008/0396
72712 [796] are applied.

72713 **NAME**

72714 wcstoumax ¶convert a wide-character string to an integer type

72715 **SYNOPSIS**

72716 #include <stddef.h>

72717 #include <inttypes.h>

72718 uintmax_t wcstoumax(const wchar_t *restrict *nptr*,72719 wchar_t **restrict *endptr*, int *base*);72720 **DESCRIPTION**72721 Refer to [wcstoimax\(\)](#).

72722 **NAME**

72723 wcswidth ‡number of column positions of a wide-character string

72724 **SYNOPSIS**

```
72725 XSI       #include <wchar.h>
72726       int wcswidth(const wchar_t *pwcs, size_t n);
```

72727 **DESCRIPTION**

72728 The *wcswidth()* function shall determine the number of column positions required for *n* wide-
72729 character codes (or fewer than *n* wide-character codes if a null wide-character code is
72730 encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

72731 **RETURN VALUE**

72732 The *wcswidth()* function either shall return 0 (if *pwcs* points to a null wide-character code), or
72733 return the number of column positions to be occupied by the wide-character string pointed to by
72734 *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed
72735 to by *pwcs* is not a printable wide-character code).

72736 **ERRORS**

72737 No errors are defined.

72738 **EXAMPLES**

72739 None.

72740 **APPLICATION USAGE**

72741 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the
72742 return value for a non-printable wide character is not specified.

72743 **RATIONALE**

72744 None.

72745 **FUTURE DIRECTIONS**

72746 None.

72747 **SEE ALSO**72748 [wctype\(\)](#)72749 XBD [Section 3.103](#) (on page 50), [<wchar.h>](#)72750 **CHANGE HISTORY**

72751 First released in Issue 4. Derived from the MSE working draft.

72752 **Issue 6**

72753 The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

72754 **NAME**

72755 wcsxfrm, wcsxfrm_l ‡'wide-character string transformation

72756 **SYNOPSIS**

72757 #include <wchar.h>

72758 size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,
72759 size_t n);72760 CX size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,
72761 size_t n, locale_t locale);72762 **DESCRIPTION**72763 CX For *wcsxfrm()*: The functionality described on this reference page is aligned with the ISO C
72764 standard. Any conflict between the requirements described here and the ISO C standard is
72765 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.72766 CX The *wcsxfrm()* and *wcsxfrm_l()* functions shall transform the wide-character string pointed to
72767 by *ws2* and place the resulting wide-character string into the array pointed to by *ws1*. The
72768 transformation shall be such that if *wcscmp()* is applied to two transformed wide strings, it shall
72769 CX return a value greater than, equal to, or less than 0, corresponding to the result of *wcscoll()* and
72770 *wcscoll_l()* applied to the same two original wide-character strings, and the same *LC_COLLATE*
72771 CX category of the current locale or the locale object *locale*, respectively. No more than *n* wide-
72772 character codes shall be placed into the resulting array pointed to by *ws1*, including the
72773 terminating null wide-character code. If *n* is 0, *ws1* is permitted to be a null pointer. If copying
72774 takes place between objects that overlap, the behavior is undefined.72775 CX The *wcsxfrm()* and *wcsxfrm_l()* functions shall not change the setting of *errno* if successful.72776 Since no return value is reserved to indicate an error, an application wishing to check for error
72777 situations should set *errno* to 0, then call *wcsxfrm()* or *wcsxfrm_l()*, then check *errno*.72778 The behavior is undefined if the *locale* argument to *wcsxfrm_l()* is the special locale object
72779 *LC_GLOBAL_LOCALE* or is not a valid locale object handle.72780 **RETURN VALUE**72781 CX The *wcsxfrm()* and *wcsxfrm_l()* functions shall return the length of the transformed wide-
72782 character string (not including the terminating null wide-character code). If the value returned is
72783 *n* or more, the contents of the array pointed to by *ws1* are unspecified.72784 CX On error, the *wcsxfrm()* and *wcsxfrm_l()* functions may set *errno*, but no return value is reserved
72785 to indicate an error.72786 **ERRORS**

72787 These functions may fail if:

72788 CX [EINVAL] The wide-character string pointed to by *ws2* contains wide-character codes
72789 outside the domain of the collating sequence.

72790 **EXAMPLES**

72791 None.

72792 **APPLICATION USAGE**

72793 The transformation function is such that two transformed wide-character strings can be ordered
72794 by *wscmp()* as appropriate to collating sequence information in the current locale (category
72795 *LC_COLLATE*).

72796 The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of
72797 the *ws1* array prior to making the transformation.

72798 **RATIONALE**

72799 None.

72800 **FUTURE DIRECTIONS**

72801 None.

72802 **SEE ALSO**72803 *wscmp()*, *wscoll()*72804 XBD <*wchar.h*>72805 **CHANGE HISTORY**

72806 First released in Issue 4. Derived from the MSE working draft.

72807 **Issue 5**

72808 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

72809 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.72810 **Issue 6**

72811 In earlier versions, this function was required to return -1 on error.

72812 Extensions beyond the ISO C standard are marked.

72813 The following new requirements on POSIX implementations derive from alignment with the
72814 Single UNIX Specification:

72815 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
72816 added if no conversion could be performed.

72817 The *wcsxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.72818 **Issue 7**

72819 The *wcsxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended
72820 API Set Part 4.

72821 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0728 [302], XSH/TC1-2008/0729 [283],
72822 XSH/TC1-2008/0730 [283], and XSH/TC1-2008/0731 [302] are applied.

72823 **NAME**72824 `wctob` ‡ wide-character to single-byte conversion72825 **SYNOPSIS**72826 `#include <stdio.h>`72827 `#include <wchar.h>`72828 `int wctob(wint_t c);`72829 **DESCRIPTION**

72830 CX The functionality described on this reference page is aligned with the ISO C standard. Any
72831 conflict between the requirements described here and the ISO C standard is unintentional. This
72832 volume of POSIX.1-2017 defers to the ISO C standard.

72833 The `wctob()` function shall determine whether `c` corresponds to a member of the extended
72834 character set whose character representation is a single byte when in the initial shift state.

72835 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

72836 **RETURN VALUE**

72837 The `wctob()` function shall return EOF if `c` does not correspond to a character with length one in
72838 the initial shift state. Otherwise, it shall return the single-byte representation of that character as
72839 an **unsigned char** converted to **int**.

72840 **ERRORS**

72841 No errors are defined.

72842 **EXAMPLES**

72843 None.

72844 **APPLICATION USAGE**

72845 None.

72846 **RATIONALE**

72847 None.

72848 **FUTURE DIRECTIONS**

72849 None.

72850 **SEE ALSO**72851 [btowc\(\)](#)72852 XBD [<stdio.h>](#), [<wchar.h>](#)72853 **CHANGE HISTORY**

72854 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
72855 (E).

72856 **NAME**

72857 wctomb ǂ'convert a wide-character code to a character

72858 **SYNOPSIS**

72859 #include <stdlib.h>

72860 int wctomb(char *s, wchar_t wchar);

72861 **DESCRIPTION**

72862 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 72863 conflict between the requirements described here and the ISO C standard is unintentional. This
 72864 volume of POSIX.1-2017 defers to the ISO C standard.

72865 The *wctomb()* function shall determine the number of bytes needed to represent the character
 72866 corresponding to the wide-character code whose value is *wchar* (including any change in the
 72867 shift state). It shall store the character representation (possibly multiple bytes and any special
 72868 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most
 72869 {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any
 72870 shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift
 72871 state.

72872 CX The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 72873 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 72874 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 72875 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 72876 null pointer shall cause this function to return a non-zero value if encodings have state
 72877 dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this
 72878 function to be unspecified.

72879 The *wctomb()* function need not be thread-safe.

72880 The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls
 72881 *wctomb()*.

72882 **RETURN VALUE**

72883 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,
 72884 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*
 72885 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the
 72886 number of bytes that constitute the character corresponding to the value of *wchar*.

72887 In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

72888 **ERRORS**72889 The *wctomb()* function shall fail if:

72890 CX [EILSEQ] An invalid wide-character code is detected.

72891 **EXAMPLES**

72892 None.

72893 **APPLICATION USAGE**

72894 None.

72895 **RATIONALE**

72896 None.

72897 **FUTURE DIRECTIONS**

72898 None.

72899 **SEE ALSO**72900 *mblen()*, *mbtowl()*, *mbstowcs()*, *wcstombs()*72901 XBD **<stdlib.h>**72902 **CHANGE HISTORY**

72903 First released in Issue 4. Derived from the ANSI C standard.

72904 **Issue 6**

72905 Extensions beyond the ISO C standard are marked.

72906 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

72907 **Issue 7**

72908 Austin Group Interpretations 1003.1-2001 #156 and #170 are applied.

72909 **NAME**72910 `wctrans, wctrans_l` ‡define character mapping72911 **SYNOPSIS**72912 `#include <wctype.h>`72913 `wctrans_t wctrans(const char *charclass);`72914 CX `wctrans_t wctrans_l(const char *charclass, locale_t locale);`72915 **DESCRIPTION**72916 CX For `wctrans()`: The functionality described on this reference page is aligned with the ISO C
72917 standard. Any conflict between the requirements described here and the ISO C standard is
72918 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.72919 CX The `wctrans()` and `wctrans_l()` functions are defined for valid character mapping names
72920 identified in the current locale. The `charclass` is a string identifying a generic character mapping
72921 name for which codeset-specific information is required. The following character mapping
72922 names are defined in all locales: **tolower** and **toupper**.72923 These functions shall return a value of type `wctrans_t`, which can be used as the second
72924 CX argument to subsequent calls of `towctrans()` and `towctrans_l()`.72925 CX The `wctrans()` and `wctrans_l()` functions shall determine values of `wctrans_t` according to the
72926 CX rules of the coded character set defined by character mapping information in the current locale
72927 or in the locale represented by `locale`, respectively (category `LC_CTYPE`).72928 The values returned by `wctrans()` shall be valid until a call to `setlocale()` that modifies the
72929 category `LC_CTYPE`.72930 CX The values returned by `wctrans_l()` shall be valid only in calls to `towctrans_l()` with a locale
72931 represented by `locale` with the same `LC_CTYPE` category value.72932 The behavior is undefined if the `locale` argument to `wctrans_l()` is the special locale object
72933 `LC_GLOBAL_LOCALE` or is not a valid locale object handle.72934 **RETURN VALUE**72935 CX The `wctrans()` and `wctrans_l()` functions shall return 0 and may set `errno` to indicate the error if
72936 the given character mapping name is not valid for the current locale (category `LC_CTYPE`);
72937 otherwise, they shall return a non-zero object of type `wctrans_t` that can be used in calls to
72938 CX `towctrans()` and `towctrans_l()`.72939 **ERRORS**

72940 These functions may fail if:

72941 CX **[EINVAL]** The character mapping name pointed to by `charclass` is not valid in the current
72942 locale.72943 **EXAMPLES**

72944 None.

72945 **APPLICATION USAGE**

72946 None.

72947 **RATIONALE**

72948 None.

72949 **FUTURE DIRECTIONS**

72950 None.

72951 **SEE ALSO**72952 [towctrans\(\)](#)72953 XBD [<wctype.h>](#)72954 **CHANGE HISTORY**

72955 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

72956 **Issue 7**72957 The *wctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended
72958 API Set Part 4.72959 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0732 [302], XSH/TC1-2008/0733 [289],
72960 XSH/TC1-2008/0734 [283], and XSH/TC1-2008/0735 [283] are applied.

72961 **NAME**

72962 wctype, wctype_l ‡define character class

72963 **SYNOPSIS**

72964 #include <wctype.h>

72965 wctype_t wctype(const char *property);

72966 CX wctype_t wctype_l(const char *property, locale_t locale);

72967 **DESCRIPTION**72968 CX For *wctype()*: The functionality described on this reference page is aligned with the ISO C
72969 standard. Any conflict between the requirements described here and the ISO C standard is
72970 unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.72971 CX The *wctype()* and *wctype_l()* functions are defined for valid character class names as defined in
72972 CX the current locale or in the locale represented by *locale*, respectively.72973 The *property* argument is a string identifying a generic character class for which codeset-specific
72974 type information is required. The following character class names shall be defined in all locales:

72975	alnum	digit	punct
72976	alpha	graph	space
72977	blank	lower	upper
72978	cntrl	print	xdigit

72979 Additional character class names defined in the locale definition file (category *LC_CTYPE*) can
72980 also be specified.72981 These functions shall return a value of type **wctype_t**, which can be used as the second
72982 CX argument to subsequent calls of *iswctype()* and *iswctype_l()*.72983 CX The *wctype()* and *wctype_l()* functions shall determine values of **wctype_t** according to the
72984 CX rules of the coded character set defined by character type information in the current locale or in
72985 the locale represented by *locale*, respectively (category *LC_CTYPE*).72986 The values returned by *wctype()* shall be valid until a call to *setlocale()* that modifies the category
72987 *LC_CTYPE*.72988 CX The values returned by *wctype_l()* shall be valid only in calls to *iswctype_l()* with a locale
72989 represented by *locale* with the same *LC_CTYPE* category value.72990 The behavior is undefined if the *locale* argument to *wctype_l()* is the special locale object
72991 *LC_GLOBAL_LOCALE* or is not a valid locale object handle.72992 **RETURN VALUE**72993 CX The *wctype()* and *wctype_l()* functions shall return 0 if the given character class name is not
72994 valid for the current locale (category *LC_CTYPE*); otherwise, they shall return an object of type72995 CX **wctype_t** that can be used in calls to *iswctype()* and *iswctype_l()*.72996 **ERRORS**

72997 No errors are defined.

72998 **EXAMPLES**

72999 None.

73000 **APPLICATION USAGE**

73001 None.

73002 **RATIONALE**

73003 None.

73004 **FUTURE DIRECTIONS**

73005 None.

73006 **SEE ALSO**73007 [*iswctype\(\)*](#)73008 XBD [**<wctype.h>**](#)73009 **CHANGE HISTORY**

73010 First released in Issue 4.

73011 **Issue 5**73012 The following change has been made in this version for alignment with
73013 ISO/IEC 9899:1990/Amendment 1:1995 (E):73014 The SYNOPSIS has been changed to indicate that this function and associated data types
73015 are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.73016 **Issue 7**73017 The *wctype_l()* function is added from The Open Group Technical Standard, 2006, Extended API
73018 Set Part 4.73019 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0736 [302], XSH/TC1-2008/0737 [283],
73020 and XSH/TC1-2008/0738 [283] are applied.

73021 **NAME**

73022 wctype †'number of column positions of a wide-character code

73023 **SYNOPSIS**

```
73024 XSI       #include <wctype.h>
73025           int wctype(wctype_t wc);
```

73026 **DESCRIPTION**

73027 The *wctype()* function shall determine the number of column positions required for the wide
73028 character *wc*. The application shall ensure that the value of *wc* is a character representable as a
73029 **wctype_t**, and is a wide-character code corresponding to a valid character in the current locale.

73030 **RETURN VALUE**

73031 The *wctype()* function shall either return 0 (if *wc* is a null wide-character code), or return the
73032 number of column positions to be occupied by the wide-character code *wc*, or return -1 (if *wc*
73033 does not correspond to a printable wide-character code).

73034 **ERRORS**

73035 No errors are defined.

73036 **EXAMPLES**

73037 None.

73038 **APPLICATION USAGE**

73039 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the
73040 return value for a non-printable wide character is not specified.

73041 **RATIONALE**

73042 None.

73043 **FUTURE DIRECTIONS**

73044 None.

73045 **SEE ALSO**

73046 *wcwidth()*

73047 XBD <wctype.h>

73048 **CHANGE HISTORY**

73049 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
73050 draft.

73051 **Issue 6**

73052 The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

73053 The normative text is updated to avoid use of the term “must” for application requirements.

73054 **NAME**73055 `wmemchr` ‡find a wide character in memory73056 **SYNOPSIS**73057 `#include <wchar.h>`73058 `wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);`73059 **DESCRIPTION**

73060 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73061 conflict between the requirements described here and the ISO C standard is unintentional. This
73062 volume of POSIX.1-2017 defers to the ISO C standard.

73063 The `wmemchr()` function shall locate the first occurrence of `wc` in the initial `n` wide characters of
73064 the object pointed to by `ws`. This function shall not be affected by locale and all `wchar_t` values
73065 shall be treated identically. The null wide character and `wchar_t` values not corresponding to
73066 valid characters shall not be treated specially.

73067 If `n` is zero, the application shall ensure that `ws` is a valid pointer and the function behaves as if
73068 no valid occurrence of `wc` is found.

73069 **RETURN VALUE**

73070 The `wmemchr()` function shall return a pointer to the located wide character, or a null pointer if
73071 the wide character does not occur in the object.

73072 **ERRORS**

73073 No errors are defined.

73074 **EXAMPLES**

73075 None.

73076 **APPLICATION USAGE**

73077 None.

73078 **RATIONALE**

73079 None.

73080 **FUTURE DIRECTIONS**

73081 None.

73082 **SEE ALSO**73083 [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)73084 XBD [<wchar.h>](#)73085 **CHANGE HISTORY**

73086 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
73087 (E).

73088 **Issue 6**

73089 The normative text is updated to avoid use of the term “must” for application requirements.

73090 **NAME**

73091 wmemcmp †'compare wide characters in memory

73092 **SYNOPSIS**

73093 #include <wchar.h>

73094 int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);

73095 **DESCRIPTION**73096 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73097 conflict between the requirements described here and the ISO C standard is unintentional. This
73098 volume of POSIX.1-2017 defers to the ISO C standard.73099 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by
73100 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be
73101 affected by locale and all **wchar_t** values shall be treated identically. The null wide character and
73102 **wchar_t** values not corresponding to valid characters shall not be treated specially.73103 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
73104 shall behave as if the two objects compare equal.73105 **RETURN VALUE**73106 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,
73107 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object
73108 pointed to by *ws2*.73109 **ERRORS**

73110 No errors are defined.

73111 **EXAMPLES**

73112 None.

73113 **APPLICATION USAGE**

73114 None.

73115 **RATIONALE**

73116 None.

73117 **FUTURE DIRECTIONS**

73118 None.

73119 **SEE ALSO**73120 *wmemchr()*, *wmemcpy()*, *wmemmove()*, *wmemset()*

73121 XBD <wchar.h>

73122 **CHANGE HISTORY**73123 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
73124 (E).73125 **Issue 6**

73126 The normative text is updated to avoid use of the term “must” for application requirements.

73127 **NAME**

73128 wmemcpy †'copy wide characters in memory

73129 **SYNOPSIS**

73130 #include <wchar.h>

73131 wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
73132 size_t n);73133 **DESCRIPTION**73134 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73135 conflict between the requirements described here and the ISO C standard is unintentional. This
73136 volume of POSIX.1-2017 defers to the ISO C standard.73137 The *wmemcpy()* function shall copy *n* wide characters from the object pointed to by *ws2* to the
73138 object pointed to by *ws1*. This function shall not be affected by locale and all **wchar_t** values
73139 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
73140 valid characters shall not be treated specially.73141 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
73142 shall copy zero wide characters.73143 **RETURN VALUE**73144 The *wmemcpy()* function shall return the value of *ws1*.73145 **ERRORS**

73146 No errors are defined.

73147 **EXAMPLES**

73148 None.

73149 **APPLICATION USAGE**

73150 None.

73151 **RATIONALE**

73152 None.

73153 **FUTURE DIRECTIONS**

73154 None.

73155 **SEE ALSO**73156 *wmemchr()*, *wmemcmp()*, *wmemmove()*, *wmemset()*

73157 XBD <wchar.h>

73158 **CHANGE HISTORY**73159 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
73160 (E).73161 **Issue 6**

73162 The normative text is updated to avoid use of the term “must” for application requirements.

73163 The *wmemcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

73164 **NAME**

73165 wmemmove — copy wide characters in memory with overlapping areas

73166 **SYNOPSIS**

73167 #include <wchar.h>

73168 wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);

73169 **DESCRIPTION**

73170 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73171 conflict between the requirements described here and the ISO C standard is unintentional. This
73172 volume of POSIX.1-2017 defers to the ISO C standard.

73173 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the
73174 object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object
73175 pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not
73176 overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary
73177 array are copied into the object pointed to by *ws1*.

73178 This function shall not be affected by locale and all **wchar_t** values shall be treated identically.
73179 The null wide character and **wchar_t** values not corresponding to valid characters shall not be
73180 treated specially.

73181 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
73182 shall copy zero wide characters.

73183 **RETURN VALUE**73184 The *wmemmove()* function shall return the value of *ws1*.73185 **ERRORS**

73186 No errors are defined

73187 **EXAMPLES**

73188 None.

73189 **APPLICATION USAGE**

73190 None.

73191 **RATIONALE**

73192 None.

73193 **FUTURE DIRECTIONS**

73194 None.

73195 **SEE ALSO**73196 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemset\(\)](#)73197 XBD [<wchar.h>](#)73198 **CHANGE HISTORY**

73199 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
73200 (E).

73201 **Issue 6**

73202 The normative text is updated to avoid use of the term “must” for application requirements.

73203 **NAME**73204 `wmemset` ‡set wide characters in memory73205 **SYNOPSIS**73206 `#include <wchar.h>`73207 `wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);`73208 **DESCRIPTION**

73209 CX The functionality described on this reference page is aligned with the ISO C standard. Any
73210 conflict between the requirements described here and the ISO C standard is unintentional. This
73211 volume of POSIX.1-2017 defers to the ISO C standard.

73212 The `wmemset()` function shall copy the value of `wc` into each of the first `n` wide characters of the
73213 object pointed to by `ws`. This function shall not be affected by locale and all `wchar_t` values shall
73214 be treated identically. The null wide character and `wchar_t` values not corresponding to valid
73215 characters shall not be treated specially.

73216 If `n` is zero, the application shall ensure that `ws` is a valid pointer, and the function shall copy
73217 zero wide characters.

73218 **RETURN VALUE**73219 The `wmemset()` functions shall return the value of `ws`.73220 **ERRORS**

73221 No errors are defined.

73222 **EXAMPLES**

73223 None.

73224 **APPLICATION USAGE**

73225 None.

73226 **RATIONALE**

73227 None.

73228 **FUTURE DIRECTIONS**

73229 None.

73230 **SEE ALSO**73231 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#)73232 XBD [<wchar.h>](#)73233 **CHANGE HISTORY**

73234 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
73235 (E).

73236 **Issue 6**

73237 The normative text is updated to avoid use of the term “must” for application requirements.

73238 **NAME**

73239 wordexp, wordfree — perform word expansions

73240 **SYNOPSIS**

```
73241 #include <wordexp.h>
73242 int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
73243             int flags);
73244 void wordfree(wordexp_t *pwordexp);
```

73245 **DESCRIPTION**

73246 The *wordexp()* function shall perform word expansions as described in XCU Section 2.6 (on page 2353), subject to quoting as described in XCU Section 2.2 (on page 2346), and place the list of expanded words into the structure pointed to by *pwordexp*.

73249 The *words* argument is a pointer to a string containing one or more words to be expanded. The expansions shall be the same as would be performed by the command line interpreter if *words* were the part of a command line representing the arguments to a utility. Therefore, the application shall ensure that *words* does not contain an unquoted <newline> character or any of the unquoted shell special characters '|', '&', ';', '<', '>' except in the context of command substitution as specified in XCU Section 2.6.3 (on page 2357). It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. The application shall ensure that every member of *words* which it expects to have expanded by *wordexp()* does not contain an unquoted initial comment character. The application shall also ensure that any words which it intends to be ignored (because they begin or continue a comment) are deleted from *words*. If the argument *words* contains an unquoted comment character (<number-sign>) that is the beginning of a token, *wordexp()* shall either treat the comment character as a regular character, or interpret it as a comment indicator and ignore the remainder of *words*.

73263 The structure type **wordexp_t** is defined in the <wordexp.h> header and includes at least the following members:

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp</i> <i>we_wordv</i> .

73270 The *wordexp()* function shall store the number of generated words into *pwordexp* ~~*we_wordc*~~ and a pointer to a list of pointers to words in *pwordexp* ~~*we_wordv*~~. Each individual field created during field splitting (see XCU Section 2.6.5, on page 2359) or pathname expansion (see XCU Section 2.6.6, on page 2360) shall be a separate word in the *pwordexp* ~~*we_wordv*~~ list. The words shall be in order as described in XCU Section 2.6 (on page 2353). The first pointer after the last word pointer shall be a null pointer. The expansion of special parameters described in XCU Section 2.5.2 (on page 2350) is unspecified.

73277 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()* function shall allocate other space as needed, including memory pointed to by *pwordexp* ~~*we_wordv*~~. The *wordfree()* function frees any memory associated with *pwordexp* from a previous call to *wordexp()*.

73281 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the bitwise-inclusive OR of zero or more of the following constants, which are defined in <wordexp.h>:

73283	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
73284	WRDE_DOOFFS	Make use of <i>pwordexp</i> <i>no_offs</i> . If this flag is set, <i>pwordexp</i> <i>no_offs</i> is used to specify how many null pointers to add to the beginning of <i>pwordexp</i> <i>no_wordv</i> . In other words, <i>pwordexp</i> <i>no_wordv</i> shall point to <i>pwordexp</i> <i>no_offs</i> null pointers, followed by <i>pwordexp</i> <i>no_wordc</i> word pointers, followed by a null pointer.
73285		
73286		
73287		
73288		
73289	WRDE_NOCMD	If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2017, fail if command substitution, as specified in XCU Section 2.6.3 (on page 2357), is requested.
73290		
73291		
73292	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.
73293		
73294		
73295		
73296	WRDE_SHOWERR	Do not redirect <i>stderr</i> to /dev/null .
73297	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.
73298		
73299		The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to <i>wordexp()</i> . The following rules apply to applications when two or more calls to <i>wordexp()</i> are made with the same value of <i>pwordexp</i> and without intervening calls to <i>wordfree()</i> :
73300		
73301		1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
73302		2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
73303		3. After the second and each subsequent call, <i>pwordexp</i> <i>no_wordv</i> shall point to a list containing the following:
73304		
73305		a. Zero or more null pointers, as specified by WRDE_DOOFFS and <i>pwordexp</i> <i>no_offs</i>
73306		
73307		b. Pointers to the words that were in the <i>pwordexp</i> <i>no_wordv</i> list before the call, in the same order as before
73308		
73309		c. Pointers to the new words generated by the latest call, in the specified order
73310		4. The count returned in <i>pwordexp</i> <i>no_wordc</i> shall be the total number of words from all of the calls.
73311		
73312		5. The application can change any of the fields after a call to <i>wordexp()</i> , but if it does it shall reset them to the original value before a subsequent call, using the same <i>pwordexp</i> value, to <i>wordfree()</i> or <i>wordexp()</i> with the WRDE_APPEND or WRDE_REUSE flag.
73313		
73314		
73315		If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2017, and <i>words</i> contains an unquoted character <code>␣<newline> ' '&'';' <'>' '(')' '{''}</code> —in an inappropriate context, <i>wordexp()</i> shall fail, and the number of expanded words shall be 0.
73316		
73317		
73318		
73319		Unless WRDE_SHOWERR is set in <i>flags</i> , <i>wordexp()</i> shall redirect <i>stderr</i> to /dev/null for any utilities executed as a result of command substitution while expanding <i>words</i> . If WRDE_SHOWERR is set, <i>wordexp()</i> may write messages to <i>stderr</i> if syntax errors are detected while expanding <i>words</i> , unless the <i>stderr</i> stream has wide orientation in which case the behavior is undefined. It is unspecified whether any write errors encountered while outputting such messages will affect the <i>stderr</i> error indicator or the value of <i>errno</i> .
73320		
73321		
73322		
73323		
73324		
73325		The application shall ensure that if WRDE_DOOFFS is set, then <i>pwordexp</i> <i>no_offs</i> has the same

73326 value for each *wordexp()* call and *wordfree()* call using a given *pwordexp*.

73327 The results are unspecified if WRDE_APPEND and WRDE_REUSE are both specified.

73328 The following constants are defined as error return values:

73329 WRDE_BADCHAR One of the unquoted characters `‡<newline>,'|', '&', ';', '<', '>',`
 73330 `'(', ')', '{', '}'` ‡appears in *words* in an inappropriate context.

73331 WRDE_BADVAL Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.

73332 WRDE_CMDSUB Command substitution requested when WRDE_NOCMD was set in *flags*.

73333 WRDE_NOSPACE Attempt to allocate memory failed.

73334 WRDE_SYNTAX Shell syntax error, such as unbalanced parentheses or unterminated
 73335 string.

73336 RETURN VALUE

73337 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described
 73338 in `<wordexp.h>`, shall be returned to indicate an error. If *wordexp()* returns the value
 73339 WRDE_NOSPACE, then *pwordexp* ~~we_wordc~~ and *pwordexp* ~~we_wordv~~ shall be updated to
 73340 reflect any words that were successfully expanded. In other error cases, if the WRDE_APPEND
 73341 flag was specified, *pwordexp*->*we_wordc* and *pwordexp*->*we_wordv* shall not be modified.

73342 The *wordfree()* function shall not return a value.

73343 ERRORS

73344 No errors are defined.

73345 EXAMPLES

73346 None.

73347 APPLICATION USAGE

73348 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's
 73349 expansions on a word or words obtained from a user. For example, if the application prompts
 73350 for a pathname (or list of pathnames) and then uses *wordexp()* to process the input, the user
 73351 could respond with anything that would be valid as input to the shell.

73352 The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to
 73353 prevent a user from executing shell commands. Disallowing unquoted shell special characters
 73354 also prevents unwanted side-effects, such as executing a command or writing a file.

73355 POSIX.1-2017 does not require the *wordexp()* function to be thread-safe if passed an expression
 73356 referencing an environment variable while any other thread is concurrently modifying any
 73357 environment variable; see *exec* (on page 783).

73358 Even though the WRDE_SHOWERR flag allows the implementation to write messages to *stderr*
 73359 during command substitution or syntax errors, this standard does not provide any way to detect
 73360 write failures during the output of such messages.

73361 Applications which use wide-character output functions with *stderr* should ensure that any calls
 73362 to *wordexp()* do not write to *stderr*, by avoiding use of the WRDE_SHOWERR flag.

73363 RATIONALE

73364 This function was included as an alternative to *glob()*. There had been continuing controversy
 73365 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*
 73366 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*
 73367 (which is faster, but which only performs pathname expansion, without tilde or parameter
 73368 expansion) this will satisfy the majority of applications.

73369 While *wordexp()* could be implemented entirely as a library routine, it is expected that most
73370 implementations run a shell in a subprocess to do the expansion.

73371 Two different approaches have been proposed for how the required information might be
73372 presented to the shell and the results returned. They are presented here as examples.

73373 One proposal is to extend the *echo* utility by adding a `-q` option. This option would cause *echo* to
73374 add a `<backslash>` before each `<backslash>` and `<blank>` that occurs within an argument. The
73375 *wordexp()* function could then invoke the shell as follows:

```
73376 (void) strcpy(buffer, "echo -q");
73377 (void) strcat(buffer, words);
73378 if ((flags & WRDE_SHOWERR) == 0)
73379     (void) strcat(buffer, "2>/dev/null");
73380 f = popen(buffer, "r");
```

73381 The *wordexp()* function would read the resulting output, remove unquoted `<backslash>`
73382 characters, and break into words at unquoted `<blank>` characters. If the `WRDE_NOCMD` flag
73383 was set, *wordexp()* would have to scan *words* before starting the subshell to make sure that there
73384 would be no command substitution. In any case, it would have to scan *words* for unquoted
73385 special characters.

73386 Another proposal is to add the following options to *sh*:

73387 `-w wordlist`

73388 This option provides a wordlist expansion service to applications. The words in *wordlist*
73389 shall be expanded and the following written to standard output:

- 73390 1. The count of the number of words after expansion, in decimal, followed by a null
73391 byte
- 73392 2. The number of bytes needed to represent the expanded words (not including null
73393 separators), in decimal, followed by a null byte
- 73394 3. The expanded words, each terminated by a null byte

73395 If an error is encountered during word expansion, *sh* exits with a non-zero status after
73396 writing the former to report any words successfully expanded

73397 `-P` Run in “protected” mode. If specified with the `-w` option, no command substitution shall
73398 be performed.

73399 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess
73400 using *fork()* and executing *sh* using the line:

```
73401 execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

73402 after directing standard error to `/dev/null`.

73403 It seemed objectionable for a library routine to write messages to standard error, unless explicitly
73404 requested, so *wordexp()* is required to redirect standard error to `/dev/null` to ensure that no
73405 messages are generated, even for commands executed for command substitution. The
73406 `WRDE_SHOWERR` flag can be specified to request that error messages be written.

73407 The `WRDE_REUSE` flag allows the implementation to avoid the expense of freeing and
73408 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when
73409 `WRDE_REUSE` is set.

73410 **FUTURE DIRECTIONS**

73411 None.

73412 **SEE ALSO**73413 *exec*, *fnmatch()*, *glob()*73414 XBD <[wordexp.h](#)>73415 XCU [Chapter 2](#) (on page 2345)73416 **CHANGE HISTORY**

73417 First released in Issue 4. Derived from the ISO POSIX-2 standard.

73418 **Issue 5**

73419 Moved from POSIX2 C-language Binding to BASE.

73420 **Issue 6**

73421 The normative text is updated to avoid use of the term “must” for application requirements.

73422 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the
73423 ISO/IEC 9899:1999 standard.73424 **Issue 7**

73425 Austin Group Interpretation 1003.1-2001 #148 is applied, adding APPLICATION USAGE.

73426 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0739 [460], XSH/TC1-2008/0740 [291],
73427 and XSH/TC1-2008/0741 [460] are applied.73428 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0397 [608], XSH/TC2-2008/0398 [704],
73429 XSH/TC2-2008/0399 [704], and XSH/TC2-2008/0400 [608] are applied.

73430 **NAME**

73431 wprintf ¶'print formatted wide-character output

73432 **SYNOPSIS**

73433 #include <stdio.h>

73434 #include <wchar.h>

73435 int wprintf(const wchar_t *restrict *format*, ...);73436 **DESCRIPTION**73437 Refer to *fwprintf()*.

73438 **NAME**

73439 pwrite, write ‡write on a file

73440 **SYNOPSIS**

73441 #include <unistd.h>

73442 ssize_t pwrite(int *fd*, const void **buf*, size_t *nbyte*,
73443 off_t *offset*);73444 ssize_t write(int *fd*, const void **buf*, size_t *nbyte*);73445 **DESCRIPTION**73446 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the
73447 file associated with the open file descriptor, *fd*.73448 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the
73449 *write()* function may detect and return errors as described below. In the absence of errors, or if
73450 error detection is not performed, the *write()* function shall return zero and have no other results.
73451 If *nbyte* is zero and the file is not a regular file, the results are unspecified.73452 On a regular file or other file capable of seeking, the actual writing of data shall proceed from
73453 the position in the file indicated by the file offset associated with *fd*. Before successful return
73454 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a
73455 regular file, if the position of the last byte written is greater than or equal to the length of the file,
73456 the length of the file shall be set to this position plus one.73457 On a file not capable of seeking, writing shall always take place starting at the current position.
73458 The value of a file offset associated with such a device is undefined.73459 If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file
73460 prior to each write and no intervening file modification operation shall occur between changing
73461 the file offset and the write operation.73462 XSI If a *write()* requests that more bytes be written than there is room for (for example, the file size
73463 limit of the process or the physical end of a medium), only as many bytes as there is room for
73464 shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a
73465 limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would
73466 give a failure return (except as noted below).73467 XSI If the request would cause the file size to exceed the soft file size limit for the process and there
73468 is no room for any bytes to be written, the request shall fail and the implementation shall
73469 generate the SIGXFSZ signal for the thread.73470 If *write()* is interrupted by a signal before it writes any data, it shall return -1 with *errno* set to
73471 [EINTR].73472 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the
73473 number of bytes written.73474 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.73475 After a *write()* to a regular file has successfully returned:73476 Any successful *read()* from each byte position in the file that was modified by that write
73477 shall return the data specified by the *write()* for that position until such byte positions are
73478 again modified.73479 Any subsequent successful *write()* to the same byte position in the file shall overwrite that
73480 file data.

73481 Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the
 73482 following exceptions:

73483 There is no file offset associated with a pipe, hence each write request shall append to the
 73484 end of the pipe.

73485 Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other
 73486 processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may
 73487 have data interleaved, on arbitrary boundaries, with writes by other processes, whether or
 73488 not the O_NONBLOCK flag of the file status flags is set.

73489 If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on
 73490 normal completion it shall return *nbyte*.

73491 If the O_NONBLOCK flag is set, *write()* requests shall be handled differently, in the
 73492 following ways:

73493 ‡ ~~When~~ *write()* function shall not block the thread.

73494 ‡ ~~When~~ write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there
 73495 is sufficient space available in the pipe, *write()* shall transfer all the data and return
 73496 the number of bytes requested. Otherwise, *write()* shall transfer no data and return
 73497 -1 with *errno* set to [EAGAIN].

73498 ‡ ~~When~~ write request for more than {PIPE_BUF} bytes shall cause one of the following:

73499 ‡ ~~When~~ at least one byte can be written, transfer what it can and return the
 73500 number of bytes written. When all data previously written to the pipe is read, it
 73501 shall transfer at least {PIPE_BUF} bytes.

73502 ‡ ~~When~~ no data can be written, transfer no data, and return -1 with *errno* set to
 73503 [EAGAIN].

73504 When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-
 73505 blocking writes and cannot accept the data immediately:

73506 If the O_NONBLOCK flag is clear, *write()* shall block the calling thread until the data can
 73507 be accepted.

73508 If the O_NONBLOCK flag is set, *write()* shall not block the thread. If some data can be
 73509 written without blocking the thread, *write()* shall write what it can and return the number
 73510 of bytes written. Otherwise, it shall return -1 and set *errno* to [EAGAIN].

73511 Upon successful completion, where *nbyte* is greater than 0, *write()* shall mark for update the last
 73512 data modification and last file status change timestamps of the file, and if the file is a regular file,
 73513 the S_ISUID and S_ISGID bits of the file mode may be cleared.

73514 For regular files, no data transfer shall occur past the offset maximum established in the open
 73515 file description associated with *fildev*.

73516 If *fildev* refers to a socket, *write()* shall be equivalent to *send()* with no flags set.

73517 SIO If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as
 73518 defined by synchronized I/O data integrity completion.

73519 If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as
 73520 defined by synchronized I/O file integrity completion.

73521 SHM If *fildev* refers to a shared memory object, the result of the *write()* function is unspecified.

73522	TYM	If <i>fildev</i> refers to a typed memory object, the result of the <i>write()</i> function is unspecified.
73523	OB XSR	If <i>fildev</i> refers to a STREAM, the operation of <i>write()</i> shall be determined by the values of the minimum and maximum <i>nbyte</i> range (packet size) accepted by the STREAM. These values are determined by the topmost STREAM module. If <i>nbyte</i> falls within the packet size range, <i>nbyte</i> bytes shall be written. If <i>nbyte</i> does not fall within the range and the minimum packet size value is 0, <i>write()</i> shall break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If <i>nbyte</i> does not fall within the range and the minimum value is non-zero, <i>write()</i> shall fail with <i>errno</i> set to [ERANGE]. Writing a zero-length buffer (<i>nbyte</i> is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue <code>I_SWROPT ioctl()</code> to enable zero-length messages to be sent across the pipe or FIFO.
73524		
73525		
73526		
73527		
73528		
73529		
73530		
73531		
73532		
73533		
73534		When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:
73535		
73536		If <code>O_NONBLOCK</code> is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), <i>write()</i> shall block until data can be accepted.
73537		
73538		
73539		If <code>O_NONBLOCK</code> is set and the STREAM cannot accept data, <i>write()</i> shall return <code>-1</code> and set <i>errno</i> to [EAGAIN].
73540		
73541		If <code>O_NONBLOCK</code> is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, <i>write()</i> shall terminate and return the number of bytes written.
73542		
73543		
73544		In addition, <i>write()</i> shall fail if the STREAM head has processed an asynchronous error before the call. In this case, the value of <i>errno</i> does not reflect the result of <i>write()</i> , but reflects the prior error.
73545		
73546		
73547		The <i>pwrite()</i> function shall be equivalent to <i>write()</i> , except that it writes into a given position and does not change the file offset (regardless of whether <code>O_APPEND</code> is set). The first three arguments to <i>pwrite()</i> are the same as <i>write()</i> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. An attempt to perform a <i>pwrite()</i> on a file that is incapable of seeking shall result in an error.
73548		
73549		
73550		
73551		
73552		RETURN VALUE
73553		Upon successful completion, these functions shall return the number of bytes actually written to the file associated with <i>fildev</i> . This number shall never be greater than <i>nbyte</i> . Otherwise, <code>-1</code> shall be returned and <i>errno</i> set to indicate the error.
73554		
73555		
73556		ERRORS
73557		These functions shall fail if:
73558		[EAGAIN] The file is neither a pipe, nor a FIFO, nor a socket, the <code>O_NONBLOCK</code> flag is set for the file descriptor, and the thread would be delayed in the <i>write()</i> operation.
73559		
73560		
73561		[EBADF] The <i>fildev</i> argument is not a valid file descriptor open for writing.
73562		[EFBIG] An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written.
73563	XSI	
73564		

73565		[EFBIG]	The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fildev</i> .
73566			
73567			
73568		[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
73569			
73570		[EIO]	The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.
73571			
73572			
73573			
73574			
73575		[ENOSPC]	There was no free space remaining on the device containing the file.
73576	OB XSR	[ERANGE]	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .
73577			
73578			The <i>write()</i> function shall fail if:
73579		[EINVAL]	The file is a regular file or block special file, and the <i>offset</i> argument is negative. The file offset shall remain unchanged.
73580			
73581		[ESPIPE]	The file is incapable of seeking.
73582			The <i>write()</i> function shall fail if:
73583		[EAGAIN]	The file is a pipe or FIFO, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the write operation.
73584			
73585		[EAGAIN] or [EWOULDBLOCK]	
73586			The file is a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the write operation.
73587			
73588		[ECONNRESET]	A write was attempted on a socket that is not connected.
73589		[EPIPE]	An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.
73590			
73591			
73592		[EPIPE]	A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, a SIGPIPE signal shall also be sent to the thread.
73593			
73594			
73595			These functions may fail if:
73596	OB XSR	[EINVAL]	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
73597			
73598		[EIO]	A physical I/O error has occurred.
73599		[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
73600		[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
73601			
73602	OB XSR	[ENXIO]	A hangup occurred on the STREAM being written to.
73603	OB XSR		A write to a STREAMS file may fail if an error message has been received at the STREAM head.
73604			In this case, <i>errno</i> is set to the value included in the error message.

- 73605 The *write()* function may fail if:
- 73606 [EACCES] A write was attempted on a socket and the calling process does not have
73607 appropriate privileges.
- 73608 [ENETDOWN] A write was attempted on a socket and the local network interface used to
73609 reach the destination is down.
- 73610 [ENETUNREACH]
73611 A write was attempted on a socket and no route to the network is present.

73612 EXAMPLES

73613 Writing from a Buffer

73614 The following example writes data from the buffer pointed to by *buf* to the file associated with
73615 the file descriptor *fd*.

```
73616 #include <sys/types.h>
73617 #include <string.h>
73618 ...
73619 char buf[20];
73620 size_t nbytes;
73621 ssize_t bytes_written;
73622 int fd;
73623 ...
73624 strcpy(buf, "This is a test\n");
73625 nbytes = strlen(buf);
73626 bytes_written = write(fd, buf, nbytes);
73627 ...
```

73628 APPLICATION USAGE

73629 None.

73630 RATIONALE

73631 See also the RATIONALE section in *read()*.

73632 An attempt to write to a pipe or FIFO has several major characteristics:

73633 *Atomic/non-atomic:* A write is atomic if the whole amount written in one operation is not
73634 interleaved with data from any other process. This is useful when there are multiple
73635 writers sending data to a single reader. Applications need to know how large a write
73636 request can be expected to be performed atomically. This maximum is called {PIPE_BUF}.
73637 This volume of POSIX.1-2017 does not say whether write requests for more than
73638 {PIPE_BUF} bytes are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall
73639 be atomic.

73640 *Blocking/immediate:* Blocking is only possible with O_NONBLOCK clear. If there is enough
73641 space for all the data requested to be written immediately, the implementation should do
73642 so. Otherwise, the calling thread may block; that is, pause until enough space is available
73643 for writing. The effective size of a pipe or FIFO (the maximum amount that can be written
73644 in one operation without blocking) may vary dynamically, depending on the
73645 implementation, so it is not possible to specify a fixed value for it.

73646 *Complete/partial/deferred:* A write request:

```
73647 int fildes;
73648 size_t nbyte;
```

```

73649     ssize_t ret;
73650     char *buf;

73651     ret = write(fildev, buf, nbytes);

```

73652 may return:

73653 Complete *ret*=*nbyte*

73654 Partial *ret*<*nbyte*

73655 This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with
73656 *nbyte*>{PIPE_BUF}), this volume of POSIX.1-2017 does not guarantee
73657 atomicity, even if *ret*≤{PIPE_BUF}, because atomicity is guaranteed according
73658 to the amount *requested*, not the amount *written*.

73659 Deferred: *ret*=−1, *errno*=[EAGAIN]

73660 This error indicates that a later request may succeed. It does not indicate that
73661 it *shall* succeed, even if *nbyte*≤{PIPE_BUF}, because if no process reads from
73662 the pipe or FIFO, the write never succeeds. An application could usefully
73663 count the number of times [EAGAIN] is caused by a particular value of
73664 *nbyte*>{PIPE_BUF} and perhaps do later writes with a smaller value, on the
73665 assumption that the effective size of the pipe may have decreased.

73666 Partial and deferred writes are only possible with O_NONBLOCK set.

73667 The relations of these properties are shown in the following tables:

Write to a Pipe or FIFO with O_NONBLOCK clear			
Immediately Writable:	None	Some	<i>nbyte</i>
<i>nbyte</i> ≤{PIPE_BUF}	Atomic blocking <i>nbyte</i>	Atomic blocking <i>nbyte</i>	Atomic immediate <i>nbyte</i>
<i>nbyte</i> >{PIPE_BUF}	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>

73673 If the O_NONBLOCK flag is clear, a write request shall block if the amount writable
73674 immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall never
73675 block.

Write to a Pipe or FIFO with O_NONBLOCK set			
Immediately Writable:	None	Some	<i>nbyte</i>
<i>nbyte</i> ≤{PIPE_BUF}	−1, [EAGAIN]	−1, [EAGAIN]	Atomic <i>nbyte</i>
<i>nbyte</i> >{PIPE_BUF}	−1, [EAGAIN]	< <i>nbyte</i> or −1, [EAGAIN]	≤ <i>nbyte</i> or −1, [EAGAIN]

73681 There is no exception regarding partial writes when O_NONBLOCK is set. With the exception
73682 of writing to an empty pipe, this volume of POSIX.1-2017 does not specify exactly when a partial
73683 write is performed since that would require specifying internal details of the implementation.
73684 Every application should be prepared to handle partial writes when O_NONBLOCK is set and
73685 the requested amount is greater than {PIPE_BUF}, just as every application should be prepared
73686 to handle partial writes on other kinds of file descriptors.

73687 The intent of forcing writing at least one byte if any can be written is to assure that each write
73688 makes progress if there is any room in the pipe. If the pipe is empty, {PIPE_BUF} bytes must be
73689 written; if not, at least some progress must have been made.

73690 Where this volume of POSIX.1-2017 requires −1 to be returned and *errno* set to [EAGAIN], most

73691 historical implementations return zero (with the O_NDELAY flag set, which is the historical
 73692 predecessor of O_NONBLOCK, but is not itself in this volume of POSIX.1-2017). The error
 73693 indications in this volume of POSIX.1-2017 were chosen so that an application can distinguish
 73694 these cases from end-of-file. While *write()* cannot receive an indication of end-of-file, *read()* can,
 73695 and the two functions have similar return values. Also, some existing systems (for example,
 73696 Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file
 73697 indication; for those systems, a return value of zero from *write()* indicates a successful write of
 73698 an end-of-file indication.

73699 Implementations are allowed, but not required, to perform error checking for *write()* requests of
 73700 zero bytes.

73701 The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be
 73702 written to a pipe in a single operation) was considered, but rejected, because this concept would
 73703 unnecessarily limit application writing.

73704 See also the discussion of O_NONBLOCK in *read()*.

73705 Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be
 73706 proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the
 73707 calls are made by different processes. A similar requirement applies to multiple write operations
 73708 to the same file position. This is needed to guarantee the propagation of data from *write()* calls
 73709 to subsequent *read()* calls. This requirement is particularly significant for networked file
 73710 systems, where some caching schemes violate these semantics.

73711 Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writew()*
 73712 also obey these semantics. A new “high-performance” write analog that did not follow these
 73713 serialization requirements would also be permitted by this wording. This volume of
 73714 POSIX.1-2017 is also silent about any effects of application-level caching (such as that done by
 73715 *stdio*).

73716 This volume of POSIX.1-2017 does not specify the value of the file offset after an error is
 73717 returned; there are too many cases. For programming errors, such as [EBADF], the concept is
 73718 meaningless since no file is involved. For errors that are detected immediately, such as
 73719 [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however,
 73720 an updated value would be very useful and is the behavior of many implementations.

73721 This volume of POSIX.1-2017 does not specify the behavior of concurrent writes to a regular file
 73722 from multiple threads, except that each write is atomic (see [Section 2.9.7](#), on page 522).
 73723 Applications should use some form of concurrency control.

73724 This volume of POSIX.1-2017 intentionally does not specify any *pwrite()* errors related to pipes,
 73725 FIFOs, and sockets other than [ESPIPE].

73726 FUTURE DIRECTIONS

73727 None.

73728 SEE ALSO

73729 [*chmod\(\)*](#), [*creat\(\)*](#), [*dup\(\)*](#), [*fcntl\(\)*](#), [*getrlimit\(\)*](#), [*lseek\(\)*](#), [*open\(\)*](#), [*pipe\(\)*](#), [*read\(\)*](#), [*ulimit\(\)*](#), [*writew\(\)*](#)

73730 XBD [*<limits.h>*](#), [*<stropts.h>*](#), [*<sys/uio.h>*](#), [*<unistd.h>*](#)

73731 CHANGE HISTORY

73732 First released in Issue 1. Derived from Issue 1 of the SVID.

73733 **Issue 5**

73734 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
73735 Threads Extension.

73736 Large File Summit extensions are added.

73737 The *pwrite()* function is added.

73738 **Issue 6**

73739 The DESCRIPTION states that the *write()* function does not block the thread. Previously this
73740 said “process” rather than “thread”.

73741 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
73742 marked as part of the XSI STREAMS Option Group.

73743 The following new requirements on POSIX implementations derive from alignment with the
73744 Single UNIX Specification:

73745 The DESCRIPTION now states that if *write()* is interrupted by a signal after it has
73746 successfully written some data, it returns the number of bytes written. In the POSIX.1-1988
73747 standard, it was optional whether *write()* returned the number of bytes written, or whether
73748 it returned -1 with *errno* set to [EINTR]. This is a FIPS requirement.

73749 The following changes are made to support large files:

73750 † For regular files, no data transfer occurs past the offset maximum established in the
73751 open file description associated with the *files*.

73752 † A second [EFBIG] error condition is added.

73753 The [EIO] error condition is added.

73754 The [EPIPE] error condition is added for when a pipe has only one end open.

73755 The [ENXIO] optional error condition is added.

73756 Text referring to sockets is added to the DESCRIPTION.

73757 The following changes were made to align with the IEEE P1003.1a draft standard:

73758 The effect of reading zero bytes is clarified.

73759 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
73760 *write()* results are unspecified for typed memory objects.

73761 The following error conditions are added for operations on sockets: [EAGAIN],
73762 [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].

73763 The [EIO] error is made optional.

73764 The [ENOBUFFS] error is added for sockets.

73765 The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN],
73766 and [ENETUNREACH].

73767 The *writen()* function is split out into a separate reference page.

73768 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/146 is applied, updating text in the
73769 ERRORS section from “a SIGPIPE signal is generated to the calling process” to “a SIGPIPE
73770 signal shall also be sent to the thread”.

73771 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/147 is applied, making a correction to the
73772 RATIONALE.

73773 **Issue 7**

73774 The *pwrite()* function is moved from the XSI option to the Base.

73775 Functionality relating to the XSI STREAMS option is marked obsolescent.

73776 SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the
73777 *pwrite()* function, and to change the use of the phrase “file pointer” to “file offset”.

73778 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0742 [219], XSH/TC1-2008/0743 [215],
73779 XSH/TC1-2008/0744 [79], and XSH/TC1-2008/0745 [215] are applied.

73780 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0401 [676,710] and
73781 XSH/TC2-2008/0402 [966] are applied.

73782 **NAME**

73783 writev ‡write a vector

73784 **SYNOPSIS**

```
73785 XSI #include <sys/uio.h>
73786     ssize_t writev(int fildev, const struct iovec *iov, int iovcnt);
```

73787 **DESCRIPTION**

73788 The *writev()* function shall be equivalent to *write()*, except as described below. The *writev()*
 73789 function shall gather output data from the *iovcnt* buffers specified by the members of the *iov*
 73790 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than
 73791 or equal to {IOV_MAX}, as defined in <limits.h>.

73792 Each *iovec* entry specifies the base address and length of an area in memory from which data
 73793 should be written. The *writev()* function shall always write a complete area before proceeding to
 73794 the next.

73795 If *fildev* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0,
 73796 *writev()* shall return 0 and have no other effect. For other file types, the behavior is unspecified.

73797 If the sum of the *iov_len* values is greater than {SSIZE_MAX}, the operation shall fail and no data
 73798 shall be transferred.

73799 **RETURN VALUE**

73800 Upon successful completion, *writev()* shall return the number of bytes actually written.
 73801 Otherwise, it shall return a value of -1, the file-pointer shall remain unchanged, and *errno* shall
 73802 be set to indicate an error.

73803 **ERRORS**73804 Refer to *write()*.73805 In addition, the *writev()* function shall fail if:73806 [EINVAL] The sum of the *iov_len* values in the *iov* array would overflow an *ssize_t*.73807 The *writev()* function may fail and set *errno* to:73808 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.73809 **EXAMPLES**73810 **Writing Data from an Array**

73811 The following example writes data from the buffers specified by members of the *iov* array to the
 73812 file associated with the file descriptor *fd*.

```
73813 #include <sys/types.h>
73814 #include <sys/uio.h>
73815 #include <unistd.h>
73816 ...
73817 ssize_t bytes_written;
73818 int fd;
73819 char *buf0 = "short string\n";
73820 char *buf1 = "This is a longer string\n";
73821 char *buf2 = "This is the longest string in this example\n";
73822 int iovcnt;
73823 struct iovec iov[3];
```

```
73824     iov[0].iov_base = buf0;
73825     iov[0].iov_len = strlen(buf0);
73826     iov[1].iov_base = buf1;
73827     iov[1].iov_len = strlen(buf1);
73828     iov[2].iov_base = buf2;
73829     iov[2].iov_len = strlen(buf2);
73830     ...
73831     iovcnt = sizeof(iov) / sizeof(struct iovec);
73832     bytes_written = writev(fd, iov, iovcnt);
73833     ...
```

73834 APPLICATION USAGE

73835 None.

73836 RATIONALE

73837 Refer to *write()*.

73838 FUTURE DIRECTIONS

73839 None.

73840 SEE ALSO

73841 *readv()*, *write()*

73842 XBD [<limits.h>](#), [<sys/uio.h>](#)

73843 CHANGE HISTORY

73844 First released in Issue 4, Version 2.

73845 Issue 6

73846 Split out from the *write()* reference page.

73847 **NAME**

73848 wscanf ¶convert formatted wide-character input

73849 **SYNOPSIS**

73850 #include <stdio.h>

73851 #include <wchar.h>

73852 int wscanf(const wchar_t *restrict *format*, ...);73853 **DESCRIPTION**73854 Refer to *fwscanf()*.

73855 **NAME**73856 `y0, y1, yn` ‡Bessel functions of the second kind73857 **SYNOPSIS**

```
73858 XSI      #include <math.h>
73859         double y0(double x);
73860         double y1(double x);
73861         double yn(int n, double x);
```

73862 **DESCRIPTION**

73863 The `y0()`, `y1()`, and `yn()` functions shall compute Bessel functions of x of the second kind of
 73864 orders 0, 1, and n , respectively.

73865 An application wishing to check for error situations should set `errno` to zero and call
 73866 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 73867 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 73868 zero, an error has occurred.

73869 **RETURN VALUE**

73870 Upon successful completion, these functions shall return the relevant Bessel value of x of the
 73871 second kind.

73872 MXX If x is NaN, NaN shall be returned.

73873 If the x argument to these functions is negative, `-HUGE_VAL` or NaN shall be returned, and a
 73874 domain error may occur.

73875 If x is 0.0, `-HUGE_VAL` shall be returned and a pole error may occur.

73876 If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

73877 If the correct result would cause overflow, `-HUGE_VAL` or 0.0 shall be returned and a range
 73878 error may occur.

73879 **ERRORS**

73880 These functions may fail if:

73881 Domain Error The value of x is negative.

73882 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 73883 then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling`
 73884 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception
 73885 shall be raised.

73886 Pole Error The value of x is zero.

73887 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 73888 then `errno` shall be set to [ERANGE]. If the integer expression
 73889 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the divide-by-zero
 73890 floating-point exception shall be raised.

73891 Range Error The correct result would cause overflow.

73892 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 73893 then `errno` shall be set to [ERANGE]. If the integer expression
 73894 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the overflow
 73895 floating-point exception shall be raised.

73896 Range Error The value of x is too large in magnitude, or the correct result would cause
73897 underflow.

73898 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
73899 then *errno* shall be set to [ERANGE]. If the integer expression
73900 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
73901 floating-point exception shall be raised.

73902 EXAMPLES

73903 None.

73904 APPLICATION USAGE

73905 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
73906 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

73907 RATIONALE

73908 None.

73909 FUTURE DIRECTIONS

73910 None.

73911 SEE ALSO

73912 [*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*isnan\(\)*](#), [*j0\(\)*](#)

73913 XBD Section 4.20 (on page 117), [<math.h>](#)

73914 CHANGE HISTORY

73915 First released in Issue 1. Derived from Issue 1 of the SVID.

73916 Issue 5

73917 The DESCRIPTION is updated to indicate how an application should check for an error. This
73918 text was previously published in the APPLICATION USAGE section.

73919 Issue 6

73920 The normative text is updated to avoid use of the term “must” for application requirements.

73921 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
73922 with the ISO/IEC 9899:1999 standard.

73923 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/148 is applied, updating the RETURN
73924 VALUE and ERRORS sections. The changes are made for consistency with the general rules
73925 stated in “Treatment of Error Conditions for Mathematical Functions” in the Base Definitions
73926 volume of POSIX.1-2017.

73927 Issue 7

73928 POSIX.1-2008, Technical Corrigendum 1, XSH/TC1-2008/0746 [68] is applied.

73929

 *Open Group Standard*

73930

Vol. 3:

73931

Shell and Utilities, Issue 7

73932

The Open Group

73933

The Institute of Electrical and Electronics Engineers, Inc.

73934

73935

Introduction

73936

The Shell and Utilities volume of POSIX.1-2017 describes the commands and utilities offered to application programs by POSIX-conformant systems.

73937

1.1 Relationship to Other Documents

73938

1.1.1 System Interfaces

73939

73940

This subsection describes some of the features provided by the System Interfaces volume of POSIX.1-2017 that are assumed to be globally available on all systems conforming to this volume of POSIX.1-2017. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of POSIX.1-2017 that are required by all of the utilities defined in this volume of POSIX.1-2017; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

73941

73942

73943

73944

73945

73946

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of POSIX.1-2017. Utility and function description statements override these defaults when appropriate.

73947

73948

1.1.1.1 Process Attributes

73949

73950

The following process attributes, as described in the System Interfaces volume of POSIX.1-2017, are assumed to be supported for all processes in this volume of POSIX.1-2017:

73951

73952

Controlling Terminal	Real Group ID
Current Working Directory	Real User ID
Effective Group ID	Root Directory
Effective User ID	Saved Set-Group-ID
File Descriptors	Saved Set-User-ID
File Mode Creation Mask	Session Membership
Process Group ID	Supplementary Group IDs
Process ID	

73953

73954

73955

73956

73957

73958

73959

A conforming implementation may include additional process attributes.

73960

1.1.1.2 Concurrent Execution of Processes

73961

73962

The following functionality of the *fork()* function defined in the System Interfaces volume of POSIX.1-2017 shall be available on all systems conforming to this volume of POSIX.1-2017:

73963

73964

1. Independent processes shall be capable of executing independently without either process terminating.

73965

73966 2. A process shall be able to create a new process with all of the attributes referenced in
 73967 [Section 1.1.1.1](#) (on page 2327), determined according to the semantics of a call to the *fork()*
 73968 function defined in the System Interfaces volume of POSIX.1-2017 followed by a call in
 73969 the child process to one of the *exec* functions defined in the System Interfaces volume of
 73970 POSIX.1-2017.

73971 1.1.1.3 *File Access Permissions*

73972 The file access control mechanism described by XBD [Section 4.5](#) (on page 108) shall apply to all
 73973 files on an implementation conforming to this volume of POSIX.1-2017.

73974 1.1.1.4 *File Read, Write, and Creation*

73975 If a file that does not exist is to be written, it shall be created as described below, unless the
 73976 utility description states otherwise.

73977 When a file that does not exist is created, the following features defined in the System Interfaces
 73978 volume of POSIX.1-2017 shall apply unless the utility or function description states otherwise:

- 73979 1. The user ID of the file shall be set to the effective user ID of the calling process.
- 73980 2. The group ID of the file shall be set to the effective group ID of the calling process or the
 73981 group ID of the directory in which the file is being created.

- 73982 3. If the file is a regular file, the permission bits of the file shall be set to:

73983 S_IROTH | S_IWOTH | S_IRGRP | S_IWGRP | S_IRUSR | S_IWUSR

73984 (see the description of *File Modes* in XBD [Chapter 13](#) (on page 219), `<sys/stat.h>`) except
 73985 that the bits specified by the file mode creation mask of the process shall be cleared. If the
 73986 file is a directory, the permission bits shall be set to:

73987 S_IRWXU | S_IRWXG | S_IRWXO

73988 except that the bits specified by the file mode creation mask of the process shall be
 73989 cleared.

- 73990 4. The last data access, last data modification, and last file status change timestamps of the
 73991 file shall be updated as specified in XBD [Section 4.9](#) (on page 109).

- 73992 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length
 73993 zero.

- 73994 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS}
 73995 variable is in effect for the directory in which the symbolic link would be created.

- 73996 7. Unless otherwise specified, the file created shall be a regular file.

73997 When an attempt is made to create a file that already exists, the utility shall take the action
 73998 indicated in [Table 1-1](#) (on page 2329) corresponding to the type of the file the utility is trying to
 73999 create and the type of the existing file, unless the utility description states otherwise.

74000

Table 1-1 Actions when Creating a File that Already Exists

74001

74002

74003

74004

74005

74006

74007

74008

74009

74010

74011

74012

74013

74014

Existing Type	New Type											Function Creating New	
	B	C	D	F	L	M	P	Q	R	S	T		
A <i>fattach()</i> -ed STREAM	F	F	F	F	F	‡	'	—	OF	—	U	N/A	
B Block Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **	
C Character Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **	
D Directory	F	F	F	F	F	‡	'	‡	'	‡	'	<i>mkdir()</i> U	
F FIFO Special File	F	F	F	F	F	‡	'	‡	'	‡	'	<i>mkfifo()</i> U	
L Symbolic Link	F	F	F	F	F	‡	'	‡	'	L	‡	'	<i>symlink()</i>
M Shared Memory	F	F	F	F	F	‡	'	‡	'	‡	'	<i>shm_open()</i> U	
P Semaphore	F	F	F	F	F	‡	'	‡	'	‡	'	<i>sem_open()</i> U	
Q Message Queue	F	F	F	F	F	‡	'	‡	'	‡	'	<i>mq_open()</i> U	
R Regular File	F	F	F	F	F	‡	'	‡	'	F	‡	'	<i>open()</i>
S Socket	F	F	F	F	F	‡	'	‡	'	‡	'	<i>bind()</i> ‡ ' U	
T Typed Memory	F	F	F	F	F	U	U	U	U	U	U	*	

74015

The following codes are used in [Table 1-1](#):

74016

74017

74018

F Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.

74019

74020

74021

74022

FL Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

74023

74024

O Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

74025

74026

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.

74027

74028

2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

74029

OF The named file shall be opened with the consequences defined for that file type.

74030

74031

74032

RF Regular file. When attempting to create a regular file, and the existing file is a regular file:

1. The user ID, group ID, and permission bits of the file shall not be changed.

2. The file shall be truncated to zero length.

74033

74034

3. The last data modification and last file status change timestamps shall be marked for update.

74035

‡ The effect is implementation-defined unless specified by the utility description.

74036

U The effect is unspecified unless specified by the utility description.

74037

***** There is no portable way to create a file of this type.

74038

****** Not portable.

74039

74040

74041

When a file is to be appended, the file shall be opened in a manner equivalent to using the `O_APPEND` flag, without the `O_TRUNC` flag, in the `open()` function defined in the System Interfaces volume of POSIX.1-2017.

74042 When a file is to be read or written, the file shall be opened with an access mode corresponding
 74043 to the operation to be performed. If file access permissions deny access, the requested operation
 74044 shall fail.

74045 1.1.1.5 File Removal

74046 When a directory that is the root directory or current working directory of any process is
 74047 removed, the effect is implementation-defined. If file access permissions deny access, the
 74048 requested operation shall fail. Otherwise, when a file is removed:

- 74049 1. Its directory entry shall be removed from the file system.
- 74050 2. The link count of the file shall be decremented.
- 74051 3. If the file is an empty directory (see XBD [Section 3.144](#), on page 56):
 - 74052 a. If no process has the directory open, the space occupied by the directory shall be
 74053 freed and the directory shall no longer be accessible.
 - 74054 b. If one or more processes have the directory open, the directory contents shall be
 74055 preserved until all references to the file have been closed.
- 74056 4. If the file is a directory that is not empty, the last file status change timestamp shall be
 74057 marked for update.
- 74058 5. If the file is not a directory:
 - 74059 a. If the link count becomes zero:
 - 74060 i. If no process has the file open, the space occupied by the file shall be freed
 74061 and the file shall no longer be accessible.
 - 74062 ii. If one or more processes have the file open, the file contents shall be
 74063 preserved until all references to the file have been closed.
 - 74064 b. If the link count is not reduced to zero, the last file status change timestamp shall
 74065 be marked for update.
- 74066 6. The last data modification and last file status change timestamps of the containing
 74067 directory shall be marked for update.

74068 1.1.1.6 File Time Values

74069 All files shall have the three time values described by XBD [Section 4.9](#) (on page 109).

74070 1.1.1.7 File Contents

74071 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of
 74072 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the
 74073 following operations defined in the System Interfaces volume of POSIX.1-2017:

```
74074 while (read (fildes, buf, nbytes) > 0)
74075     ;
```

74076 If the file is indicated by a pathname *pathname*, the file descriptor shall be determined by the
 74077 equivalent of the following operation defined in the System Interfaces volume of POSIX.1-2017:

```
74078 fildes = open (pathname, O_RDONLY);
```

74079 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data

74080 returned by *read()* would vary with different values, the value shall be one that results in the
74081 most data being returned.

74082 If the *read()* function calls would return an error, it is unspecified whether the contents of the file
74083 are considered to include any data from offsets in the file beyond where the error would be
74084 returned.

74085 1.1.1.8 Pathname Resolution

74086 The pathname resolution algorithm, described by XBD Section 4.13 (on page 111), shall be used
74087 by implementations conforming to this volume of POSIX.1-2017; see also XBD Section 4.6 (on
74088 page 109).

74089 1.1.1.9 Changing the Current Working Directory

74090 When the current working directory (see XBD Section 3.122, on page 53) is to be changed, unless
74091 the utility or function description states otherwise, the operation shall succeed unless a call to
74092 the *chdir()* function defined in the System Interfaces volume of POSIX.1-2017 would fail when
74093 invoked with the new working directory pathname as its argument.

74094 1.1.1.10 Establish the Locale

74095 The functionality of the *setlocale()* function defined in the System Interfaces volume of
74096 POSIX.1-2017 shall be available on all systems conforming to this volume of POSIX.1-2017; that
74097 is, utilities that require the capability of establishing an international operating environment
74098 shall be permitted to set the specified category of the international environment.

74099 1.1.1.11 Actions Equivalent to Functions

74100 Some utility descriptions specify that a utility performs actions equivalent to a function defined
74101 in the System Interfaces volume of POSIX.1-2017. Such specifications require only that the
74102 external effects be equivalent, not that any effect within the utility and visible only to the utility
74103 be equivalent.

74104 1.1.2 Concepts Derived from the ISO C Standard

74105 Some of the standard utilities perform complex data manipulation using their own procedure
74106 and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS
74107 sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type
74108 conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard,
74109 as described in the following sections. Note that there is no requirement that the standard
74110 utilities be implemented in any particular programming language.

74111 1.1.2.1 Arithmetic Precision and Operations

74112 Integer variables and constants, including the values of operands and option-arguments, used
74113 by the standard utilities listed in this volume of POSIX.1-2017 shall be implemented as
74114 equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as
74115 equivalent to the ISO C standard **double** type. Conversions between types shall be as described
74116 in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned
74117 by the input to the application.

74118 Arithmetic operators and control flow keywords shall be implemented as equivalent to those in
 74119 the cited ISO C standard section, as listed in [Table 1-2](#).

74120 **Table 1-2** Selected ISO C Standard Operators and Control Flow Keywords

Operation	ISO C Standard Equivalent Reference
()	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! sizeof()	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
== !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
	Section 6.5.14, Logical OR Operator
expr?expr:expr	Section 6.5.15, Conditional Operator
=, *=, /=, %= <<=, >>=, &=, ^=, =	Section 6.5.16, Assignment Operators
if () if () ... else switch ()	Section 6.8.4, Selection Statements
while () do ... while () for ()	Section 6.8.5, Iteration Statements
goto continue break return	Section 6.8.6, Jump Statements

74161 The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5,
74162 Expressions, of the ISO C standard.

74163 1.1.2.2 *Mathematical Functions*

74164 Any mathematical functions with the same names as those in the following sections of the ISO C
74165 standard:

74166 Section 7.12, Mathematics, <math.h>

74167 Section 7.20.2, Pseudo-Random Sequence Generation Functions

74168 shall be implemented to return the results equivalent to those returned from a call to the
74169 corresponding function described in the ISO C standard.

74170 1.2 Utility Limits

74171 This section lists magnitude limitations imposed by a specific implementation. The braces
74172 notation, {LIMIT}, is used in this volume of POSIX.1-2017 to indicate these values, but the braces
74173 are not part of the name.

74174 **Table 1-3** Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the border_start keyword in XBD Section 7.3.2 (on page 147).	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	2 048
{POSIX_RE_DUP_MAX}	Maximum number of repeated occurrences of a BRE or ERE interval expression; see XBD Section 9.3.6 (on page 186) and Section 9.4.6 (on page 190).	255

74200 The values specified in [Table 1-3](#) represent the lowest values conforming implementations shall
74201 provide and, consequently, the largest values on which an application can rely without further

74202 enquiries, as described below. These values shall be accessible to applications via the *getconf*
74203 utility (see *getconf*, on page 2831).

74204 Implementations may provide more liberal, or less restrictive, values than shown in [Table 1-3](#)
74205 (on page 2333). These possibly more liberal values are accessible using the symbols in [Table 1-4](#).

74206 The *sysconf()* function defined in the System Interfaces volume of POSIX.1-2017 or the *getconf*
74207 utility return the value of each symbol on each specific implementation. The value so retrieved is
74208 the largest, or most liberal, value that is available throughout the session lifetime, as determined
74209 at session creation. The literal names shown in the table apply only to the *getconf* utility; the
74210 high-level language binding describes the exact form of each name to be used by the interfaces
74211 in that binding.

74212 All numeric limits defined by the System Interfaces volume of POSIX.1-2017, such as
74213 {PATH_MAX}, shall also apply to this volume of POSIX.1-2017. All the utilities defined by this
74214 volume of POSIX.1-2017 are implicitly limited by these values, unless otherwise noted in the
74215 utility descriptions.

74216 It is not guaranteed that the application can actually reach the specified limit of an
74217 implementation in any given case, or at all, as a lack of virtual memory or other resources may
74218 prevent this. The limit value indicates only that the implementation does not specifically impose
74219 any arbitrary, more restrictive limit.

74220 **Table 1-4** Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the order_start keyword in XBD Section 7.3.2 (on page 147).	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}

74244	Name	Description	Minimum Value
74245	{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}
74246			
74247		Maximum number of repeated occurrences of a BRE or ERE interval expression; see XBD Section 9.3.6 (on page 186) and Section 9.4.6 (on page 190).	{POSIX_RE_DUP_MAX}
74248			
74249			
74250			
74251			
74252			
74253			
74254	{RE_DUP_MAX}		
74255			
74256			
74257			
74258			
74259			

74260 The following value may be a constant within an implementation or may vary from one
74261 pathname to another.

74262 {POSIX2_SYMLINKS}

74263 When referring to a directory, the system supports the creation of symbolic links within that
74264 directory; for non-directory files, the meaning of {POSIX2_SYMLINKS} is undefined.

74265 1.3 Grammar Conventions

74266 Portions of this volume of POSIX.1-2017 are expressed in terms of a special grammar notation. It
74267 is used to portray the complex syntax of certain program input. The grammar is based on the
74268 syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input,
74269 suitable for program use; the lexical processing and all semantic requirements are described only
74270 in textual form. The grammar is not based on source used in any traditional implementation and
74271 has not been tested with the semantic code that would normally be required to accompany it.
74272 Furthermore, there is no implication that the partial *yacc* code presented represents the most
74273 efficient, or only, means of supporting the complex syntax within the utility. Implementations
74274 may use other programming languages or algorithms, as long as the syntax supported is the
74275 same as that represented by the grammar.

74276 The following typographical conventions are used in the grammar; they have no significance
74277 except to aid in reading.

74278 The identifiers for the reserved words of the language are shown with a leading capital
74279 letter. (These are terminals in the grammar; for example, **While**, **Case**.)

74280 The identifiers for terminals in the grammar are all named with uppercase letters and
74281 underscores; for example, **NEWLINE**, **ASSIGN_OP**, **NAME**.

74282 The identifiers for non-terminals are all lowercase.

74283 1.4 Utility Description Defaults

74284 This section describes all of the subsections used within the utility descriptions, including:

74285 Intended usage of the section

74286 Global defaults that affect all the standard utilities

74287 The meanings of notations used in this volume of POSIX.1-2017 that are specific to
74288 individual utility sections

74289 **NAME**

74290 This section gives the name or names of the utility and briefly states its purpose.

74291 **SYNOPSIS**

74292 The SYNOPSIS section summarizes the syntax of the calling sequence for the utility,
74293 including options, option-arguments, and operands. Standards for utility naming are
74294 described in XBD [Section 12.2](#) (on page 216); for describing the utility's arguments in
74295 XBD [Section 12.1](#) (on page 213).

74296 **DESCRIPTION**

74297 The DESCRIPTION section describes the actions of the utility. If the utility has a very
74298 complex set of subcommands or its own procedural language, an EXTENDED
74299 DESCRIPTION section is also provided. Most explanations of optional functionality are
74300 omitted here, as they are usually explained in the OPTIONS section.

74301 As stated in [Section 1.1.1.11](#) (on page 2331), some functions are described in terms of
74302 equivalent functionality. When specific functions are cited, the implementation shall
74303 provide equivalent functionality including side-effects associated with successful
74304 execution of the function. The treatment of errors and intermediate results from the
74305 individual functions cited is generally not specified by this volume of POSIX.1-2017.
74306 See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all
74307 actions associated with errors encountered by the utility.

74308 **OPTIONS**

74309 The OPTIONS section describes the utility options and option-arguments, and how
74310 they modify the actions of the utility. Standard utilities that have options either fully
74311 comply with XBD [Section 12.2](#) (on page 216) or describe all deviations. Apparent
74312 disagreements between functionality descriptions in the OPTIONS and DESCRIPTION
74313 (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS
74314 section.

74315 Each OPTIONS section that uses the phrase "The ... utility shall conform to the Utility
74316 Syntax Guidelines ..." refers only to the use of the utility as specified by this volume of
74317 POSIX.1-2017; implementation extensions should also conform to the guidelines, but
74318 may allow exceptions for historical practice.

74319 Unless otherwise stated in the utility description, when given an option unrecognized
74320 by the implementation, or when a required option-argument is not provided, standard
74321 utilities shall issue a diagnostic message to standard error and exit with a non-zero exit
74322 status.

74323 All utilities in this volume of POSIX.1-2017 shall be capable of processing arguments
74324 using eight-bit transparency.

74325 **Default Behavior:** When this section is listed as "None.", it means that the
74326 implementation need not support any options. Standard utilities that do not accept
74327 options, but that do accept operands, shall recognize "--" as a first argument to be
74328 discarded.

74329 The requirement for recognizing "--" is because conforming applications need a way
 74330 to shield their operands from any arbitrary options that the implementation may
 74331 provide as an extension. For example, if the standard utility *foo* is listed as taking no
 74332 options, and the application needed to give it a pathname with a leading <hyphen-
 74333 minus>, it could safely do it as:

```
74334 foo -- -myfile
```

74335 and avoid any problems with **-m** used as an extension.

74336 OPERANDS

74337 The OPERANDS section describes the utility operands, and how they affect the actions
 74338 of the utility. Apparent disagreements between functionality descriptions in the
 74339 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be
 74340 resolved in favor of the OPERANDS section.

74341 If an operand naming a file can be specified as '-', which means to use the standard
 74342 input instead of a named file, this is explicitly stated in this section. Unless otherwise
 74343 stated, the use of multiple instances of '-' to mean standard input in a single
 74344 command produces unspecified results.

74345 Unless otherwise stated, the standard utilities that accept operands shall process those
 74346 operands in the order specified in the command line.

74347 **Default Behavior:** When this section is listed as ``None.'', it means that the
 74348 implementation need not support any operands.

74349 STDIN

74350 The STDIN section describes the standard input of the utility. This section is frequently
 74351 merely a reference to the following section, as many utilities treat standard input and
 74352 input files in the same manner. Unless otherwise stated, all restrictions described in the
 74353 INPUT FILES section shall apply to this section as well.

74354 Use of a terminal for standard input can cause any of the standard utilities that read
 74355 standard input to stop when used in the background. For this reason, applications
 74356 should not use interactive features in scripts to be placed in the background.

74357 The specified standard input format of the standard utilities shall not depend on the
 74358 existence or value of the environment variables defined in this volume of POSIX.1-2017,
 74359 except as provided by this volume of POSIX.1-2017.

74360 **Default Behavior:** When this section is listed as ``Not used.'', it means that the standard
 74361 input shall not be read when the utility is used as described by this volume of
 74362 POSIX.1-2017.

74363 INPUT FILES

74364 The INPUT FILES section describes the files, other than the standard input, used as
 74365 input by the utility. It includes files named as operands and option-arguments as well
 74366 as other files that are referred to, such as start-up and initialization files, databases, and
 74367 so on. Commonly-used files are generally described in one place and cross-referenced
 74368 by other utilities.

74369 All utilities in this volume of POSIX.1-2017 shall be capable of processing input files
 74370 using eight-bit transparency.

74371 When a standard utility reads a seekable input file and terminates without an error
 74372 before it reaches end-of-file, the utility shall ensure that the file offset in the open file
 74373 description is properly positioned just past the last byte processed by the utility. For
 74374 files that are not seekable, the state of the file offset in the open file description for that

74375 file is unspecified. A conforming application shall not assume that the following three
74376 commands are equivalent:

```
74377 tail -n +2 file
74378 (sed -n 1q; cat) < file
74379 cat file | (sed -n 1q; cat)
```

74380 The second command is equivalent to the first only when the file is seekable. The third
74381 command leaves the file offset in the open file description in an unspecified state. Other
74382 utilities, such as *head*, *read*, and *sh*, have similar properties.

74383 Some of the standard utilities, such as filters, process input files a line or a block at a
74384 time and have no restrictions on the maximum input file size. Some utilities may have
74385 size limitations that are not as obvious as file space or memory limitations. Such
74386 limitations should reflect resource limitations of some sort, not arbitrary limits set by
74387 implementors. Implementations shall document those utilities that are limited by
74388 constraints other than file system space, available memory, and other limits specifically
74389 cited by this volume of POSIX.1-2017, and identify what the constraint is and indicate a
74390 way of estimating when the constraint would be reached. Similarly, some utilities
74391 descend the directory tree (recursively). Implementations shall also document any
74392 limits that they may have in descending the directory tree that are beyond limits cited
74393 by this volume of POSIX.1-2017.

74394 When an input file is described as a “text file”, the utility produces undefined results if
74395 given input that is not from a text file, unless otherwise stated. Some utilities (for
74396 example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline>
74397 convention; unless otherwise stated, the utility need not be able to accumulate more
74398 than {LINE_MAX} bytes from a set of multiple, continued input lines. Thus, for a
74399 conforming application the total of all the continued lines in a set cannot exceed
74400 {LINE_MAX}. If a utility using the escaped <newline> convention detects an end-of-
74401 file condition immediately after an escaped <newline>, the results are unspecified.

74402 Record formats are described in a notation similar to that used by the C-language
74403 function, *printf()*. See XBD [Chapter 5](#) (on page 121) for a description of this notation.
74404 The format description is intended to be sufficiently rigorous to allow other
74405 applications to generate these input files. However, since <blank>s can legitimately be
74406 included in some of the fields described by the standard utilities, particularly in locales
74407 other than the POSIX locale, this intent is not always realized.

74408 **Default Behavior:** When this section is listed as “None.”, it means that no input files
74409 are required to be supplied when the utility is used as described by this volume of
74410 POSIX.1-2017.

74411 ENVIRONMENT VARIABLES

74412 The ENVIRONMENT VARIABLES section lists what variables affect the utility’s
74413 execution.

74414 The entire manner in which environment variables described in this volume of
74415 POSIX.1-2017 affect the behavior of each utility is described in the ENVIRONMENT
74416 VARIABLES section for that utility, in conjunction with the global effects of the *LANG*,
74417 XSI *LC_ALL*, and *NLSPATH* environment variables described in XBD [Chapter 8](#) (on page
74418 173). The existence or value of environment variables described in this volume of
74419 POSIX.1-2017 shall not otherwise affect the specified behavior of the standard utilities.
74420 Any effects of the existence or value of environment variables not described by this
74421 volume of POSIX.1-2017 upon the standard utilities are unspecified.

74422 For those standard utilities that use environment variables as a means for selecting a

74423 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to
74424 the path search described for *PATH* in XBD [Chapter 8](#) (on page 173).

74425 All utilities in this volume of POSIX.1-2017 shall be capable of processing environment
74426 variable names and values using eight-bit transparency.

74427 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of
74428 the utility is not directly affected by environment variables described by this volume of
74429 POSIX.1-2017 when the utility is used as described by this volume of POSIX.1-2017.

74430 ASYNCHRONOUS EVENTS

74431 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as
74432 signals and what signals are caught.

74433 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard
74434 action for all other signals; see [Section 1.4](#) (on page 2336)” it means that the action taken
74435 as a result of the signal shall be one of the following:

- 74436 1. The action shall be that inherited from the parent according to the rules of
74437 inheritance of signal actions defined in the System Interfaces volume of
74438 POSIX.1-2017.
- 74439 2. When no action has been taken to change the default, the default action shall be
74440 that specified by the System Interfaces volume of POSIX.1-2017.
- 74441 3. The result of the utility’s execution is as if default actions had been taken.

74442 A utility is permitted to catch a signal, perform some additional processing (such as
74443 deleting temporary files), restore the default signal action (or action inherited from the
74444 parent process), and resignal itself.

74445 STDOUT

74446 The STDOUT section completely describes the standard output of the utility. This
74447 section is frequently merely a reference to the following section, OUTPUT FILES,
74448 because many utilities treat standard output and output files in the same manner.

74449 Use of a terminal for standard output may cause any of the standard utilities that write
74450 standard output to stop when used in the background. For this reason, applications
74451 should not use interactive features in scripts to be placed in the background.

74452 Record formats are described in a notation similar to that used by the C-language
74453 function, *printf()*. See XBD [Chapter 5](#) (on page 121) for a description of this notation.

74454 The specified standard output of the standard utilities shall not depend on the
74455 existence or value of the environment variables defined in this volume of POSIX.1-2017,
74456 except as provided by this volume of POSIX.1-2017.

74457 Some of the standard utilities describe their output using the verb *display*, defined in
74458 XBD [Section 3.133](#) (on page 54). Output described in the STDOUT sections of such
74459 utilities may be produced using means other than standard output. When standard
74460 output is directed to a terminal, the output described shall be written directly to the
74461 terminal. Otherwise, the results are undefined.

74462 **Default Behavior:** When this section is listed as “Not used.”, it means that the standard
74463 output shall not be written when the utility is used as described by this volume of
74464 POSIX.1-2017.

74465 STDERR

74466 The STDERR section describes the standard error output of the utility. Only those
74467 messages that are purposely sent by the utility are described.

74468 Use of a terminal for standard error may cause any of the standard utilities that write
 74469 standard error output to stop when used in the background. For this reason,
 74470 applications should not use interactive features in scripts to be placed in the
 74471 background.

74472 The format of diagnostic messages for most utilities is unspecified, but the language
 74473 and cultural conventions of diagnostic and informative messages whose format is
 74474 unspecified by this volume of POSIX.1-2017 should be affected by the setting of
 74475 XSI `LC_MESSAGES` and `NLSPATH`.

74476 The specified standard error output of standard utilities shall not depend on the
 74477 existence or value of the environment variables defined in this volume of POSIX.1-2017,
 74478 except as provided by this volume of POSIX.1-2017.

74479 **Default Behavior:** When this section is listed as “The standard error shall be used only
 74480 for diagnostic messages.”, it means that, unless otherwise stated, the diagnostic
 74481 messages shall be sent to the standard error only when the exit status indicates that an
 74482 error occurred and the utility is used as described by this volume of POSIX.1-2017.

74483 When this section is listed as “Not used.”, it means that the standard error shall not be
 74484 used when the utility is used as described in this volume of POSIX.1-2017.

74485 OUTPUT FILES

74486 The OUTPUT FILES section completely describes the files created or modified by the
 74487 utility. Temporary or system files that are created for internal usage by this utility or
 74488 other parts of the implementation (for example, spool, log, and audit files) are not
 74489 described in this, or any, section. The utilities creating such files and the names of such
 74490 files are unspecified. If applications are written to use temporary or intermediate files,
 74491 they should use the `TMPDIR` environment variable, if it is set and represents an
 74492 accessible directory, to select the location of temporary files.

74493 Implementations shall ensure that temporary files, when used by the standard utilities,
 74494 are named so that different utilities or multiple instances of the same utility can operate
 74495 simultaneously without regard to their working directories, or any other process
 74496 characteristic other than process ID. There are two exceptions to this rule:

- 74497 1. Resources for temporary files other than the name space (for example, disk
 74498 space, available directory entries, or number of processes allowed) are not
 74499 guaranteed.
- 74500 2. Certain standard utilities generate output files that are intended as input for
 74501 other utilities (for example, `lex` generates `lex.yy.c`), and these cannot have unique
 74502 names. These cases are explicitly identified in the descriptions of the respective
 74503 utilities.

74504 Any temporary file created by the implementation shall be removed by the
 74505 implementation upon a utility’s successful exit, exit because of errors, or before
 74506 termination by any of the `SIGHUP`, `SIGINT`, or `SIGTERM` signals, unless specified
 74507 otherwise by the utility description.

74508 Receipt of the `SIGQUIT` signal should generally cause termination (unless in some
 74509 debugging mode) that would bypass any attempted recovery actions.

74510 Record formats are described in a notation similar to that used by the C-language
 74511 function, `printf()`; see XBD [Chapter 5](#) (on page 121) for a description of this notation.

74512 **Default Behavior:** When this section is listed as “None.”, it means that no files are
 74513 created or modified as a consequence of direct action on the part of the utility when the
 74514 utility is used as described by this volume of POSIX.1-2017. However, the utility may

74515 create or modify system files, such as log files, that are outside the utility's normal
74516 execution environment.

74517 EXTENDED DESCRIPTION

74518 The EXTENDED DESCRIPTION section provides a place for describing the actions of
74519 very complicated utilities, such as text editors or language processors, which typically
74520 have elaborate command languages.

74521 **Default Behavior:** When this section is listed as "None.", no further description is
74522 necessary.

74523 EXIT STATUS

74524 The EXIT STATUS section describes the values the utility shall return to the calling
74525 program, or shell, and the conditions that cause these values to be returned. Usually,
74526 utilities return zero for successful completion and values greater than zero for various
74527 error conditions. If specific numeric values are listed in this section, the system shall
74528 use those values for the errors described. In some cases, status values are listed more
74529 loosely, such as >0. A strictly conforming application shall not rely on any specific
74530 value in the range shown and shall be prepared to receive any value in the range.

74531 For example, a utility may list zero as a successful return, 1 as a failure for a specific
74532 reason, and >1 as "an error occurred". In this case, unspecified conditions may cause a
74533 2 or 3, or other value, to be returned. A conforming application should be written so
74534 that it tests for successful exit status values (zero in this case), rather than relying upon
74535 the single specific error value listed in this volume of POSIX.1-2017. In that way, it has
74536 maximum portability, even on implementations with extensions.

74537 Unspecified error conditions may be represented by specific values not listed in this
74538 volume of POSIX.1-2017.

74539 CONSEQUENCES OF ERRORS

74540 The CONSEQUENCES OF ERRORS section describes the effects on the environment,
74541 file systems, process state, and so on, when error conditions occur. It does not describe
74542 error messages produced or exit status values used.

74543 The many reasons for failure of a utility are generally not specified by the utility
74544 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of
74545 options, arguments, or environment variables; invalid usage of the complex syntaxes
74546 expressed in EXTENDED DESCRIPTION sections; resource exhaustion; difficulties
74547 accessing, creating, reading, or writing files; or difficulties associated with the
74548 privileges of the process.

74549 The following shall apply to each utility, unless otherwise stated:

74550 If the requested action cannot be performed on an operand representing a file,
74551 directory, user, process, and so on, the utility shall issue a diagnostic message to
74552 standard error and continue processing the next operand in sequence, but the
74553 final exit status shall be returned as non-zero.

74554 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if
74555 the requested action cannot be performed on a file or directory encountered in the
74556 hierarchy, the utility shall issue a diagnostic message to standard error and
74557 continue processing the remaining files in the hierarchy, but the final exit status
74558 shall be returned as non-zero.

74559 If the requested action characterized by an option or option-argument cannot be
74560 performed, the utility shall issue a diagnostic message to standard error and the
74561 exit status returned shall be non-zero.

74562 When an unrecoverable error condition is encountered, the utility shall exit with a
74563 non-zero exit status.

74564 A diagnostic message shall be written to standard error whenever an error
74565 condition occurs.

74566 When a utility encounters an error condition several actions are possible, depending on
74567 the severity of the error and the state of the utility. Included in the possible actions of
74568 various utilities are: deletion of temporary or intermediate work files; deletion of
74569 incomplete files; validity checking of the file system or directory.

74570 **Default Behavior:** When this section is listed as “Default.”, it means that any changes
74571 to the environment, file systems, process state, and so on are unspecified.

74572 **APPLICATION USAGE**

74573 This section is informative.

74574 The APPLICATION USAGE section gives advice to the application programmer or
74575 user about the way the utility should be used.

74576 **EXAMPLES**

74577 This section is informative.

74578 The EXAMPLES section gives one or more examples of usage, where appropriate. In
74579 the event of conflict between an example and a normative part of the specification, the
74580 normative material is to be taken as correct.

74581 In all examples, quoting has been used, showing how sample commands (utility names
74582 combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to
74583 the *system()* function defined in the System Interfaces volume of POSIX.1-2017. Such
74584 quoting would not be used if the utility is invoked using one of the *exec* functions
74585 defined in the System Interfaces volume of POSIX.1-2017.

74586 **RATIONALE**

74587 This section is informative.

74588 This section contains historical information concerning the contents of this volume of
74589 POSIX.1-2017 and why features were included or discarded by the standard
74590 developers.

74591 **FUTURE DIRECTIONS**

74592 This section is informative.

74593 The FUTURE DIRECTIONS section should be used as a guide to current thinking; there
74594 is not necessarily a commitment to implement all of these future directions in their
74595 entirety.

74596 **SEE ALSO**

74597 This section is informative.

74598 The SEE ALSO section lists related entries.

74599 **CHANGE HISTORY**

74600 This section is informative.

74601 This section shows the derivation of the entry and any significant changes that have
74602 been made to it.

74603 Certain of the standard utilities describe how they can invoke other utilities or applications, such
74604 as by passing a command string to the command interpreter. The external influences (STDIN,
74605 ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF

74606 ERRORS, and so on) of such invoked utilities are not described in the section concerning the
74607 standard utility that invokes them.

74608 1.5 Considerations for Utilities in Support of Files of Arbitrary Size

74609 The following utilities support files of any size up to the maximum that can be created by the
74610 implementation. This support includes correct writing of file size-related values (such as file
74611 sizes and offsets, line numbers, and block counts) and correct interpretation of command line
74612 arguments that contain such values.

74613	<i>basename</i>	Return non-directory portion of pathname.
74614	<i>cat</i>	Concatenate and print files.
74615	<i>cd</i>	Change working directory.
74616	<i>chgrp</i>	Change file group ownership.
74617	<i>chmod</i>	Change file modes.
74618	<i>chown</i>	Change file ownership.
74619	<i>cksum</i>	Write file checksums and sizes.
74620	<i>cmp</i>	Compare two files.
74621	<i>cp</i>	Copy files.
74622	<i>dd</i>	Convert and copy a file.
74623	<i>df</i>	Report free disk space.
74624	<i>dirname</i>	Return directory portion of pathname.
74625	<i>du</i>	Estimate file space usage.
74626	<i>find</i>	Find files.
74627	<i>ln</i>	Link files.
74628	<i>ls</i>	List directory contents.
74629	<i>mkdir</i>	Make directories.
74630	<i>mv</i>	Move files.
74631	<i>pathchk</i>	Check pathnames.
74632	<i>pwd</i>	Return working directory name.
74633	<i>rm</i>	Remove directory entries.
74634	<i>rmdir</i>	Remove directories.
74635	<i>sh</i>	Shell, the standard command language interpreter.
74636	<i>sum</i>	Print checksum and block or byte count of a file.
74637	<i>test</i>	Evaluate expression.
74638	<i>touch</i>	Change file access and modification times.
74639	<i>ulimit</i>	Set or report file size limit.

74640 Exceptions to the requirement that utilities support files of any size up to the maximum are as
 74641 follows:

- 74642 1. Uses of files as command scripts, or for configuration or control, are exempt. For example,
 74643 it is not required that *sh* be able to read an arbitrarily large **.profile**.
- 74644 2. Shell input and output redirection are exempt. For example, it is not required that the
 74645 redirections *sum < file* or *echo foo > file* succeed for an arbitrarily large existing file.

74646 **1.6 Built-In Utilities**

74647 Any of the standard utilities may be implemented as regular built-in utilities within the
 74648 command language interpreter. This is usually done to increase the performance of frequently
 74649 used utilities or to achieve functionality that would be more difficult in a separate environment.
 74650 The utilities named in [Table 1-5](#) are frequently provided in built-in form. All of the utilities
 74651 named in the table have special properties in terms of command search order within the shell, as
 74652 described in [Section 2.9.1.1](#) (on page 2367).

74653 **Table 1-5 Regular Built-In Utilities**

74654	<i>alias</i>	<i>false</i>	<i>hash</i>	<i>pwd</i>	<i>ulimit</i>
74655	<i>bg</i>	<i>fc</i>	<i>jobs</i>	<i>read</i>	<i>umask</i>
74656	<i>cd</i>	<i>fg</i>	<i>kill</i>	<i>true</i>	<i>unalias</i>
74657	<i>command</i>	<i>getopts</i>	<i>newgrp</i>	<i>type</i>	<i>wait</i>

74658 However, all of the standard utilities, including the regular built-ins in the table, but not the
 74659 special built-ins described in [Section 2.14](#) (on page 2384), shall be implemented in a manner so
 74660 that they can be accessed via the *exec* family of functions as defined in the System Interfaces
 74661 volume of POSIX.1-2017 and can be invoked directly by those standard utilities that require it
 74662 (*env, find, nice, nohup, time, xargs*).

Shell Command Language

This chapter contains the definition of the Shell Command Language.

74666 2.1 Shell Introduction

The shell is a command language interpreter. This chapter describes the syntax of that command language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the System Interfaces volume of POSIX.1-2017.

The shell operates according to the following general overview of operations. The specific details are included in the cited sections of this chapter.

1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and *popen()* functions defined in the System Interfaces volume of POSIX.1-2017. If the first line of a file of shell commands starts with the characters "#!", the results are unspecified.

2. The shell breaks the input into tokens: words and operators; see [Section 2.3](#).

3. The shell parses the input into simple commands (see [Section 2.9.1](#)) and compound commands (see [Section 2.9.4](#)).

4. The shell performs various expansions (separately) on different parts of each command, resulting in a list of pathnames and fields to be treated as a command and arguments; see [Section 2.6](#).

5. The shell performs redirection (see [Section 2.7](#)) and removes redirection operators and their operands from the parameter list.

6. The shell executes a function (see [Section 2.9.5](#)), built-in (see [Section 2.14](#)), executable file, or script, giving the names of the arguments as positional parameters numbered 1 to *n*, and the name of the command (or in the case of a function within a script, the name of the script) as the positional parameter numbered 0 (see [Section 2.9.1.1](#)).

7. The shell optionally waits for the command to complete and collects the exit status (see [Section 2.8.2](#)).

74690 2.2 Quoting

74691 Quoting is used to remove the special meaning of certain characters or words to the shell.
 74692 Quoting can be used to preserve the literal meaning of the special characters in the next
 74693 paragraph, prevent reserved words from being recognized as such, and prevent parameter
 74694 expansion and command substitution within here-document processing (see [Section 2.7.4](#)).

74695 The application shall quote the following characters if they are to represent themselves:

74696 | & ; < > () \$ ` \ " ' <space> <tab> <newline>

74697 and the following may need to be quoted under certain circumstances. That is, these characters
 74698 may be special depending on conditions described elsewhere in this volume of POSIX.1-2017:

74699 * ? [# ~ = %

74700 The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The
 74701 here-document represents another form of quoting; see [Section 2.7.4](#).

74702 2.2.1 Escape Character (Backslash)

74703 A <backslash> that is not quoted shall preserve the literal value of the following character, with
 74704 the exception of a <newline>. If a <newline> follows the <backslash>, the shell shall interpret
 74705 this as line continuation. The <backslash> and <newline> shall be removed before splitting the
 74706 input into tokens. Since the escaped <newline> is removed entirely from the input and is not
 74707 replaced by any white space, it cannot serve as a token separator.

74708 2.2.2 Single-Quotes

74709 Enclosing characters in single-quotes (' ') shall preserve the literal value of each character
 74710 within the single-quotes. A single-quote cannot occur within single-quotes.

74711 2.2.3 Double-Quotes

74712 Enclosing characters in double-quotes (" ") shall preserve the literal value of all characters
 74713 within the double-quotes, with the exception of the characters backquote, <dollar-sign>, and
 74714 <backslash>, as follows:

74715 § The <dollar-sign> shall retain its special meaning introducing parameter expansion (see
 74716 [Section 2.6.2](#)), a form of command substitution (see [Section 2.6.3](#)), and arithmetic expansion
 74717 (see [Section 2.6.4](#)).

74718 The input characters within the quoted string that are also enclosed between "\$ (" and the
 74719 matching ') ' ' shall not be affected by the double-quotes, but rather shall define that
 74720 command whose output replaces the "\$ (. . .) " when the word is expanded. The
 74721 tokenizing rules in [Section 2.3](#), not including the alias substitutions in [Section 2.3.1](#), shall be
 74722 applied recursively to find the matching ') ' '.

74723 Within the string of characters from an enclosed "\$ { " to the matching ' } ' ' , an even number
 74724 of unescaped double-quotes or single-quotes, if any, shall occur. A preceding <backslash>
 74725 character shall be used to escape a literal ' { ' or ' } ' ' . The rule in [Section 2.6.2](#) shall be used
 74726 to determine the matching ' } ' ' .

74727 ``` The backquote shall retain its special meaning introducing the other form of command
 74728 substitution (see [Section 2.6.3](#)). The portion of the quoted string from the initial backquote
 74729 and the characters up to the next backquote that is not preceded by a `<backslash>`, having
 74730 escape characters removed, defines that command whose output replaces "``...``" when
 74731 the word is expanded. Either of the following cases produces undefined results:

74732 A single-quoted or double-quoted string that begins, but does not end, within the
 74733 "``...``" sequence

74734 A "``...``" sequence that begins, but does not end, within the same double-quoted
 74735 string

74736 `\` The `<backslash>` shall retain its special meaning as an escape character (see [Section 2.2.1](#))
 74737 only when followed by one of the following characters when considered special:

74738 `$` ``` `"` `\` `<newline>`

74739 The application shall ensure that a double-quote is preceded by a `<backslash>` to be included
 74740 within double-quotes. The parameter `'@'` has special meaning inside double-quotes and is
 74741 described in [Section 2.5.2](#).

74742 2.3 Token Recognition

74743 The shell shall read its input in terms of lines. (For details about how the shell reads its input, see
 74744 the description of *sh*.) The input lines can be of unlimited length. These lines shall be parsed
 74745 using two major modes: ordinary token recognition and processing of here-documents.

74746 When an `io_here` token has been recognized by the grammar (see [Section 2.10](#)), one or more of
 74747 the subsequent lines immediately following the next `NEWLINE` token form the body of one or
 74748 more here-documents and shall be parsed according to the rules of [Section 2.7.4](#).

74749 When it is not processing an `io_here`, the shell shall break its input into tokens by applying the
 74750 first applicable rule below to the next character in its input. The token shall be from the current
 74751 position in the input until a token is delimited according to one of the rules below; the characters
 74752 forming the token are exactly those in the input, including any quoting characters. If it is
 74753 indicated that a token is delimited, and no characters have been included in a token, processing
 74754 shall continue until an actual token is delimited.

- 74755 1. If the end of input is recognized, the current token (if any) shall be delimited.
- 74756 2. If the previous character was used as part of an operator and the current character is not
 74757 quoted and can be used with the previous characters to form an operator, it shall be used
 74758 as part of that (operator) token.
- 74759 3. If the previous character was used as part of an operator and the current character cannot
 74760 be used with the previous characters to form an operator, the operator containing the
 74761 previous character shall be delimited.
- 74762 4. If the current character is `<backslash>`, single-quote, or double-quote and it is not quoted,
 74763 it shall affect quoting for subsequent characters up to the end of the quoted text. The rules
 74764 for quoting are as described in [Section 2.2](#). During token recognition no substitutions
 74765 shall be actually performed, and the result token shall contain exactly the characters that
 74766 appear in the input (except for `<newline>` joining), unmodified, including any embedded
 74767 or enclosing quotes or substitution operators, between the `<quotation-mark>` and the end
 74768 of the quoted text. The token shall not be delimited by the end of the quoted field.

- 74769 5. If the current character is an unquoted '\$' or '`', the shell shall identify the start of any
 74770 candidates for parameter expansion (Section 2.6.2), command substitution (Section 2.6.3),
 74771 or arithmetic expansion (Section 2.6.4) from their introductory unquoted character
 74772 sequences: '\$' or "\${", "\$(" or '`', and "\$(", respectively. The shell shall read
 74773 sufficient input to determine the end of the unit to be expanded (as explained in the cited
 74774 sections). While processing the characters, if instances of expansions or quoting are
 74775 found nested within the substitution, the shell shall recursively process them in the
 74776 manner specified for the construct that is found. The characters found from the
 74777 beginning of the substitution to its end, allowing for any recursion necessary to recognize
 74778 embedded constructs, shall be included unmodified in the result token, including any
 74779 embedded or enclosing substitution operators or quotes. The token shall not be delimited
 74780 by the end of the substitution.
- 74781 6. If the current character is not quoted and can be used as the first character of a new
 74782 operator, the current token (if any) shall be delimited. The current character shall be used
 74783 as the beginning of the next (operator) token.
- 74784 7. If the current character is an unquoted <blank>, any token containing the previous
 74785 character is delimited and the current character shall be discarded.
- 74786 8. If the previous character was part of a word, the current character shall be appended to
 74787 that word.
- 74788 9. If the current character is a '#', it and all subsequent characters up to, but excluding, the
 74789 next <newline> shall be discarded as a comment. The <newline> that ends the line is not
 74790 considered part of the comment.
- 74791 10. The current character is used as the start of a new word.

74792 Once a token is delimited, it is categorized as required by the grammar in Section 2.10.

74793 2.3.1 Alias Substitution

74794 After a token has been delimited, but before applying the grammatical rules in Section 2.10, a
 74795 resulting word that is identified to be the command name word of a simple command shall be
 74796 examined to determine whether it is an unquoted, valid alias name. However, reserved words in
 74797 correct grammatical context shall not be candidates for alias substitution. A valid alias name (see
 74798 XBD Section 3.10) shall be one that has been defined by the *alias* utility and not subsequently
 74799 undefined using *unalias*. Implementations also may provide predefined valid aliases that are in
 74800 effect when the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not
 74801 currently processing an alias of the same name, the word shall be replaced by the value of the
 74802 alias; otherwise, it shall not be replaced.

74803 If the value of the alias replacing the word ends in a <blank>, the shell shall check the next
 74804 command word for alias substitution; this process shall continue until a word is found that is
 74805 not a valid alias or an alias value does not end in a <blank>.

74806 When used as specified by this volume of POSIX.1-2017, alias definitions shall not be inherited
 74807 by separate invocations of the shell or by the utility execution environments invoked by the
 74808 shell; see Section 2.12.

74809 2.4 Reserved Words

74810 Reserved words are words that have special meaning to the shell; see [Section 2.9](#). The following
74811 words shall be recognized as reserved words:

74812	!	do	esac	in
74813	{	done	fi	then
74814	}	elif	for	until
74815	case	else	if	while

74816 This recognition shall only occur when none of the characters is quoted and when the word is
74817 used as:

74818 The first word of a command

74819 The first word following one of the reserved words other than **case**, **for**, or **in**

74820 The third word in a **case** command (only **in** is valid in this case)

74821 The third word in a **for** command (only **in** and **do** are valid in this case)

74822 See the grammar in [Section 2.10](#).

74823 The following words may be recognized as reserved words on some implementations (when
74824 none of the characters are quoted), causing unspecified results:

74825	[[]]	function	select
-------	-----------	-----------	-----------------	---------------

74826 Words that are the concatenation of a name and a <colon> (' : ') are reserved; their use produces
74827 unspecified results.

74828 2.5 Parameters and Variables

74829 A parameter can be denoted by a name, a number, or one of the special characters listed in
74830 [Section 2.5.2](#). A variable is a parameter denoted by a name.

74831 A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can
74832 only be unset by using the *unset* special built-in command.

74833 2.5.1 Positional Parameters

74834 A positional parameter is a parameter denoted by the decimal value represented by one or more
74835 digits, other than the single digit 0. The digits denoting the positional parameters shall always
74836 be interpreted as a decimal value, even if there is a leading zero. When a positional parameter
74837 with more than one digit is specified, the application shall enclose the digits in braces (see
74838 [Section 2.6.2](#)). Positional parameters are initially assigned when the shell is invoked (see *sh*),
74839 temporarily replaced when a shell function is invoked (see [Section 2.9.5](#)), and can be reassigned
74840 with the *set* special built-in command.

74841 **2.5.2 Special Parameters**

74842 Listed below are the special parameters and the values to which they shall expand. Only the
 74843 values of the special parameters are listed; see [Section 2.6](#) for a detailed summary of all the
 74844 stages involved in expanding words.

74845 @ Expands to the positional parameters, starting from one, initially producing one field for
 74846 each positional parameter that is set. When the expansion occurs in a context where field
 74847 splitting will be performed, any empty fields may be discarded and each of the non-empty
 74848 fields shall be further split as described in [Section 2.6.5](#). When the expansion occurs within
 74849 double-quotes, the behavior is unspecified unless one of the following is true:

74850 Field splitting as described in [Section 2.6.5](#) would be performed if the expansion were
 74851 not within double-quotes (regardless of whether field splitting would have any effect;
 74852 for example, if *IFS* is null).

74853 The double-quotes are within the *word* of a $\${parameter:-word}$ or a $\${parameter:+word}$
 74854 expansion (with or without the `<colon>`; see [Section 2.6.2](#)) which would have been
 74855 subject to field splitting if *parameter* had been expanded instead of *word*.

74856 If one of these conditions is true, the initial fields shall be retained as separate fields, except
 74857 that if the parameter being expanded was embedded within a word, the first field shall be
 74858 joined with the beginning part of the original word and the last field shall be joined with the
 74859 end part of the original word. In all other contexts the results of the expansion are
 74860 unspecified. If there are no positional parameters, the expansion of '@' shall generate zero
 74861 fields, even when '@' is within double-quotes; however, if the expansion is embedded
 74862 within a word which contains one or more other parts that expand to a quoted null string,
 74863 these null string(s) shall still produce an empty field, except that if the other parts are all
 74864 within the same double-quotes as the '@', it is unspecified whether the result is zero fields
 74865 or one empty field.

74866 * Expands to the positional parameters, starting from one, initially producing one field for
 74867 each positional parameter that is set. When the expansion occurs in a context where field
 74868 splitting will be performed, any empty fields may be discarded and each of the non-empty
 74869 fields shall be further split as described in [Section 2.6.5](#). When the expansion occurs in a
 74870 context where field splitting will not be performed, the initial fields shall be joined to form a
 74871 single field with the value of each parameter separated by the first character of the *IFS*
 74872 variable if *IFS* contains at least one character, or separated by a `<space>` if *IFS* is unset, or
 74873 with no separation if *IFS* is set to a null string.

74874 # Expands to the decimal number of positional parameters. The command name (parameter
 74875 0) shall not be counted in the number given by '#' because it is a special parameter, not a
 74876 positional parameter.

74877 ? Expands to the decimal exit status of the most recent pipeline (see [Section 2.9.2](#)).

74878 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated
 74879 into a string) as specified on invocation, by the *set* special built-in command, or implicitly
 74880 by the shell.

74881 \$ Expands to the decimal process ID of the invoked shell. In a subshell (see [Section 2.12](#)), '\$'
 74882 shall expand to the same value as that of the current shell.

74883 ! Expands to the decimal process ID of the most recent background command (see [Section](#)
 74884 [2.9.3](#)) executed from the current shell. (For example, background commands executed from
 74885 subshells do not affect the value of "\$!" in the current shell environment.) For a pipeline,
 74886 the process ID is that of the last command in the pipeline.

74887 0 (Zero.) Expands to the name of the shell or shell script. See *sh* for a detailed description of
 74888 how this name is derived.
 74889 See the description of the *IFS* variable in [Section 2.5.3](#).

74890 2.5.3 Shell Variables

74891 Variables shall be initialized from the environment (as defined by XBD [Chapter 8](#) and the *exec*
 74892 function in the System Interfaces volume of POSIX.1-2017) and can be given new values with
 74893 variable assignment commands. If a variable is initialized from the environment, it shall be
 74894 marked for export immediately; see the *export* special built-in. New variables can be defined and
 74895 initialized with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter
 74896 in a **for** loop, with the $\${name=word}$ expansion, or with other mechanisms provided as
 74897 implementation extensions.

74898 The following variables shall affect the execution of the shell:

74899 UP *ENV* The processing of the *ENV* shell variable shall be supported if the system
 74900 supports the User Portability Utilities option.

74901 This variable, when and only when an interactive shell is invoked, shall be
 74902 subjected to parameter expansion (see [Section 2.6.2](#)) by the shell and the
 74903 resulting value shall be used as a pathname of a file containing shell
 74904 commands to execute in the current environment. The file need not be
 74905 executable. If the expanded value of *ENV* is not an absolute pathname, the
 74906 results are unspecified. *ENV* shall be ignored if the user's real and effective
 74907 user IDs or real and effective group IDs are different.

74908 *HOME* The pathname of the user's home directory. The contents of *HOME* are used in
 74909 tilde expansion (see [Section 2.6.1](#)).

74910 *IFS* A string treated as a list of characters that is used for field splitting, expansion
 74911 of the '*' special parameter, and to split lines into fields with the *read* utility.
 74912 If the value of *IFS* includes any bytes that do not form part of a valid character,
 74913 the results of field splitting, expansion of '*', and use of the *read* utility are
 74914 unspecified.

74915 If *IFS* is not set, it shall behave as normal for an unset variable, except that
 74916 field splitting by the shell and line splitting by the *read* utility shall be
 74917 performed as if the value of *IFS* is <space><tab><newline>; see [Section 2.6.5](#).

74918 The shell shall set *IFS* to <space><tab><newline> when it is invoked.

74919 *LANG* Provide a default value for the internationalization variables that are unset or
 74920 null. (See XBD [Section 8.2](#) for the precedence of internationalization variables
 74921 used to determine the values of locale categories.)

74922 *LC_ALL* The value of this variable overrides the *LC_** variables and *LANG*, as
 74923 described in XBD [Chapter 8](#).

74924 *LC_COLLATE* Determine the behavior of range expressions, equivalence classes, and multi-
 74925 character collating elements within pattern matching.

74926 *LC_CTYPE* Determine the interpretation of sequences of bytes of text data as characters
 74927 (for example, single-byte as opposed to multi-byte characters), which
 74928 characters are defined as letters (character class **alpha**) and <blank> characters
 74929 (character class **blank**), and the behavior of character classes within pattern
 74930 matching. Changing the value of *LC_CTYPE* after the shell has started shall

74931		not affect the lexical processing of shell commands in the current shell
74932		execution environment or its subshells. Invoking a shell script or performing
74933		<code>exec sh</code> subjects the new shell to the changes in <code>LC_CTYPE</code> .
74934	<code>LC_MESSAGES</code>	Determine the language in which messages should be written.
74935	<code>LINENO</code>	Set by the shell to a decimal number representing the current sequential line
74936		number (numbered starting with 1) within a script or function before it
74937		executes each command. If the user unsets or resets <code>LINENO</code> , the variable may
74938		lose its special meaning for the life of the shell. If the shell is not currently
74939		executing a script or function, the value of <code>LINENO</code> is unspecified. This
74940		volume of POSIX.1-2017 specifies the effects of the variable only for systems
74941		supporting the User Portability Utilities option.
74942	XSI <code>NLSPATH</code>	Determine the location of message catalogs for the processing of
74943		<code>LC_MESSAGES</code> .
74944	<code>PATH</code>	A string formatted as described in XBD Chapter 8, used to effect command
74945		interpretation; see Section 2.9.1.1.
74946	<code>PPID</code>	Set by the shell to the decimal value of its parent process ID during
74947		initialization of the shell. In a subshell (see Section 2.12), <code>PPID</code> shall be set to
74948		the same value as that of the parent of the current shell. For example, <code>echo</code>
74949		<code>\$PPID</code> and <code>(echo \$PPID)</code> would produce the same value.
74950	<code>PS1</code>	Each time an interactive shell is ready to read a command, the value of this
74951		variable shall be subjected to parameter expansion and written to standard
74952		error. The default value shall be " <code>\$ </code> ". For users who have specific additional
74953		implementation-defined privileges, the default may be another,
74954		implementation-defined value. The shell shall replace each instance of the
74955		character <code>'!'</code> in <code>PS1</code> with the history file number of the next command to be
74956		typed. Escaping the <code>'!'</code> with another <code>'!'</code> (that is, <code>"!!"</code>) shall place the literal
74957		character <code>'!'</code> in the prompt. This volume of POSIX.1-2017 specifies the effects
74958		of the variable only for systems supporting the User Portability Utilities
74959		option.
74960	<code>PS2</code>	Each time the user enters a <newline> prior to completing a command line in
74961		an interactive shell, the value of this variable shall be subjected to parameter
74962		expansion and written to standard error. The default value is " <code>> </code> ". This
74963		volume of POSIX.1-2017 specifies the effects of the variable only for systems
74964		supporting the User Portability Utilities option.
74965	<code>PS4</code>	When an execution trace (<code>set -x</code>) is being performed in an interactive shell,
74966		before each line in the execution trace, the value of this variable shall be
74967		subjected to parameter expansion and written to standard error. The default
74968		value is " <code>+ </code> ". This volume of POSIX.1-2017 specifies the effects of the
74969		variable only for systems supporting the User Portability Utilities option.
74970	<code>PWD</code>	Set by the shell and by the <code>cd</code> utility. In the shell the value shall be initialized
74971		from the environment as follows. If a value for <code>PWD</code> is passed to the shell in
74972		the environment when it is executed, the value is an absolute pathname of the
74973		current working directory that is no longer than <code>{PATH_MAX}</code> bytes including
74974		the terminating null byte, and the value does not contain any components that
74975		are dot or dot-dot, then the shell shall set <code>PWD</code> to the value from the
74976		environment. Otherwise, if a value for <code>PWD</code> is passed to the shell in the
74977		environment when it is executed, the value is an absolute pathname of the
74978		current working directory, and the value does not contain any components

74979 that are dot or dot-dot, then it is unspecified whether the shell sets *PWD* to the
 74980 value from the environment or sets *PWD* to the pathname that would be
 74981 output by *pwd -P*. Otherwise, the *sh* utility sets *PWD* to the pathname that
 74982 would be output by *pwd -P*. In cases where *PWD* is set to the value from the
 74983 environment, the value can contain components that refer to files of type
 74984 symbolic link. In cases where *PWD* is set to the pathname that would be
 74985 output by *pwd -P*, if there is insufficient permission on the current working
 74986 directory, or on any parent of that directory, to determine what that pathname
 74987 would be, the value of *PWD* is unspecified. Assignments to this variable may
 74988 be ignored. If an application sets or unsets the value of *PWD*, the behaviors of
 74989 the *cd* and *pwd* utilities are unspecified.

74990 2.6 Word Expansions

74991 This section describes the various expansions that are performed on words. Not all expansions
 74992 are performed on every word, as explained in the following sections.

74993 Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and
 74994 quote removals that occur within a single word expand to a single field. It is only field splitting
 74995 or pathname expansion that can create multiple fields from a single word. The single exception
 74996 to this rule is the expansion of the special parameter '@' within double-quotes, as described in
 74997 [Section 2.5.2](#).

74998 The order of word expansion shall be as follows:

- 74999 1. Tilde expansion (see [Section 2.6.1](#)), parameter expansion (see [Section 2.6.2](#)), command
 75000 substitution (see [Section 2.6.3](#)), and arithmetic expansion (see [Section 2.6.4](#)) shall be
 75001 performed, beginning to end. See item 5 in [Section 2.3](#).
- 75002 2. Field splitting (see [Section 2.6.5](#)) shall be performed on the portions of the fields
 75003 generated by step 1, unless *IFS* is null.
- 75004 3. Pathname expansion (see [Section 2.6.6](#)) shall be performed, unless *set -f* is in effect.
- 75005 4. Quote removal (see [Section 2.6.7](#)) shall always be performed last.

75006 The expansions described in this section shall occur in the same shell environment as that in
 75007 which the command is executed.

75008 If the complete expansion appropriate for a word results in an empty field, that empty field shall
 75009 be deleted from the list of fields that form the completely expanded command, unless the
 75010 original word contained single-quote or double-quote characters.

75011 The '\$' character is used to introduce parameter expansion, command substitution, or
 75012 arithmetic evaluation. If an unquoted '\$' is followed by a character that is not one of the
 75013 following:

75014 A numeric character

75015 The name of one of the special parameters (see [Section 2.5.2](#))

75016 A valid first character of a variable name

75017 A <left-curly-bracket> (' { ')

75018 A <left-parenthesis>

75019 the result is unspecified.

75020 2.6.1 Tilde Expansion

75021 A “tilde-prefix” consists of an unquoted <tilde> character at the beginning of a word, followed
 75022 by all of the characters preceding the first unquoted <slash> in the word, or all the characters in
 75023 the word if there is no <slash>. In an assignment (see XBD Section 4.23), multiple tilde-prefixes
 75024 can be used: at the beginning of the word (that is, following the <equals-sign> of the
 75025 assignment), following any unquoted <colon>, or both. A tilde-prefix in an assignment is
 75026 terminated by the first unquoted <colon> or <slash>. If none of the characters in the tilde-prefix
 75027 are quoted, the characters in the tilde-prefix following the <tilde> are treated as a possible login
 75028 name from the user database. A portable login name cannot contain characters outside the set
 75029 given in the description of the *LOGNAME* environment variable in XBD Section 8.3. If the login
 75030 name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix is replaced by the
 75031 value of the variable *HOME*. If *HOME* is unset, the results are unspecified. Otherwise, the tilde-
 75032 prefix shall be replaced by a pathname of the initial working directory associated with the login
 75033 name obtained using the *getpwnam()* function as defined in the System Interfaces volume of
 75034 POSIX.1-2017. If the system does not recognize the login name, the results are undefined.

75035 The pathname resulting from tilde expansion shall be treated as if quoted to prevent it being
 75036 altered by field splitting and pathname expansion.

75037 2.6.2 Parameter Expansion

75038 The format for parameter expansion is as follows:

```
75039 ${expression}
```

75040 where *expression* consists of all characters until the matching '}'. Any '}' escaped by a
 75041 <backslash> or within a quoted string, and characters in embedded arithmetic expansions,
 75042 command substitutions, and variable expansions, shall not be examined in determining the
 75043 matching '}'.

75044 The simplest form for parameter expansion is:

```
75045 ${parameter}
```

75046 The value, if any, of *parameter* shall be substituted.

75047 The parameter name or symbol can be enclosed in braces, which are optional except for
 75048 positional parameters with more than one digit or when *parameter* is a name and is followed by a
 75049 character that could be interpreted as part of the name. The matching closing brace shall be
 75050 determined by counting brace levels, skipping over enclosed quoted strings, and command
 75051 substitutions.

75052 If the parameter is not enclosed in braces, and is a name, the expansion shall use the longest
 75053 valid name (see XBD Section 3.235), whether or not the variable represented by that name exists.
 75054 Otherwise, the parameter is a single-character symbol, and behavior is unspecified if that
 75055 character is neither a digit nor one of the special parameters (see Section 2.5.2).

75056 If a parameter expansion occurs inside double-quotes:

75057 Pathname expansion shall not be performed on the results of the expansion.

75058 Field splitting shall not be performed on the results of the expansion.

75059 In addition, a parameter expansion can be modified by using one of the following formats. In
 75060 each case that a value of *word* is needed (based on the state of *parameter*, as described below),
 75061 *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and
 75062 arithmetic expansion. If *word* is not needed, it shall not be expanded. The '}' character that

75063 delimits the following parameter expansion modifications shall be determined as described
75064 previously in this section and in [Section 2.2.3](#).

75065 $\${parameter}:-[word]$ **Use Default Values.** If *parameter* is unset or null, the expansion of *word*
75066 (or an empty string if *word* is omitted) shall be substituted; otherwise, the
75067 value of *parameter* shall be substituted.

75068 $\${parameter}:= [word]$ **Assign Default Values.** If *parameter* is unset or null, the expansion of
75069 *word* (or an empty string if *word* is omitted) shall be assigned to *parameter*.
75070 In all cases, the final value of *parameter* shall be substituted. Only
75071 variables, not positional parameters or special parameters, can be
75072 assigned in this way.

75073 $\${parameter}?:[word]$ **Indicate Error if Null or Unset.** If *parameter* is unset or null, the
75074 expansion of *word* (or a message indicating it is unset if *word* is omitted)
75075 shall be written to standard error and the shell exits with a non-zero exit
75076 status. Otherwise, the value of *parameter* shall be substituted. An
75077 interactive shell need not exit.

75078 $\${parameter}+:[word]$ **Use Alternative Value.** If *parameter* is unset or null, null shall be
75079 substituted; otherwise, the expansion of *word* (or an empty string if *word*
75080 is omitted) shall be substituted.

75081 In the parameter expansions shown previously, use of the <colon> in the format shall result in a
75082 test for a parameter that is unset or null; omission of the <colon> shall result in a test for a
75083 parameter that is only unset. If *parameter* is '#' and the colon is omitted, the application shall
75084 ensure that *word* is specified (this is necessary to avoid ambiguity with the string length
75085 expansion). The following table summarizes the effect of the <colon>:

	<i>parameter</i> Set and Not Null	<i>parameter</i> Set But Null	<i>parameter</i> Unset
75086 $\${parameter}:-word$	substitute <i>parameter</i>	substitute <i>word</i>	substitute <i>word</i>
75087 $\${parameter}-word$	substitute <i>parameter</i>	substitute null	substitute <i>word</i>
75088 $\${parameter}:=word$	substitute <i>parameter</i>	assign <i>word</i>	assign <i>word</i>
75089 $\${parameter}=word$	substitute <i>parameter</i>	substitute null	assign <i>word</i>
75090 $\${parameter}?word$	substitute <i>parameter</i>	error, exit	error, exit
75091 $\${parameter}?word$	substitute <i>parameter</i>	substitute null	error, exit
75092 $\${parameter}+word$	substitute <i>word</i>	substitute null	substitute null
75093 $\${parameter}+word$	substitute <i>word</i>	substitute <i>word</i>	substitute null

75096 In all cases shown with "substitute", the expression is replaced with the value shown. In all
75097 cases shown with "assign", *parameter* is assigned that value, which also replaces the expression.

75098 $\${#parameter}$ **String Length.** The length in characters of the value of *parameter* shall be
75099 substituted. If *parameter* is '*' or '@', the result of the expansion is
75100 unspecified. If *parameter* is unset and *set -u* is in effect, the expansion
75101 shall fail.

75102 The following four varieties of parameter expansion provide for substring processing. In each
75103 case, pattern matching notation (see [Section 2.13](#)), rather than regular expression notation, shall
75104 be used to evaluate the patterns. If *parameter* is '#', '*', or '@', the result of the expansion is
75105 unspecified. If *parameter* is unset and *set -u* is in effect, the expansion shall fail. Enclosing the full
75106 parameter expansion string in double-quotes shall not cause the following four varieties of
75107 pattern characters to be quoted, whereas quoting characters within the braces shall have this
75108 effect. In each variety, if *word* is omitted, the empty pattern shall be used.

75109 `${parameter%[word]}` **Remove Smallest Suffix Pattern.** The *word* shall be expanded to produce
 75110 a pattern. The parameter expansion shall then result in *parameter*, with the
 75111 smallest portion of the suffix matched by the *pattern* deleted. If present,
 75112 *word* shall not begin with an unquoted '%'.
 75113 `${parameter%%[word]}` **Remove Largest Suffix Pattern.** The *word* shall be expanded to produce a
 75114 pattern. The parameter expansion shall then result in *parameter*, with the
 75115 largest portion of the suffix matched by the *pattern* deleted.
 75116 `${parameter#[word]}` **Remove Smallest Prefix Pattern.** The *word* shall be expanded to produce
 75117 a pattern. The parameter expansion shall then result in *parameter*, with the
 75118 smallest portion of the prefix matched by the *pattern* deleted. If present,
 75119 *word* shall not begin with an unquoted '#'.
 75120 `${parameter##[word]}` **Remove Largest Prefix Pattern.** The *word* shall be expanded to produce a
 75121 pattern. The parameter expansion shall then result in *parameter*, with the
 75122 largest portion of the prefix matched by the *pattern* deleted.

75123 Examples

75124 `${parameter}`
 75125 In this example, the effects of omitting braces are demonstrated.
 75126 a=1
 75127 set 2
 75128 echo \${a}b-\$ab-\${1}0-\${10}-\${10}
 75129 **1b--20--20**
 75130 `${parameter-word}`
 75131 This example demonstrates the difference between unset and set to the empty string, as well
 75132 as the rules for finding the delimiting close brace.
 75133 foo=asdf
 75134 echo \${foo-bar}xyz}
 75135 **asdfxyz}**
 75136 foo=
 75137 echo \${foo-bar}xyz}
 75138 **xyz}**
 75139 unset foo
 75140 echo \${foo-bar}xyz}
 75141 **barxyz}**
 75142 `${parameter:-word}`
 75143 In this example, *ls* is executed only if *x* is null or unset. (The `$(ls)` command substitution
 75144 notation is explained in [Section 2.6.3](#).)
 75145 \${x:-\$(ls)}
 75146 `${parameter:=word}`
 75147 unset X
 75148 echo \${X:=abc}
 75149 **abc**
 75150 `${parameter:?word}`
 75151 unset posix
 75152 echo \${posix:?}
 75153 **sh: posix: parameter null or not set**

```

75154     ${parameter:+word}
75155         set a b c
75156         echo ${3:+posix}
75157         posix

75158     ${#parameter}
75159         HOME=/usr/posix
75160         echo ${#HOME}
75161         10

75162     ${parameter%word}
75163         x=file.c
75164         echo ${x%.c}.o
75165         file.o

75166     ${parameter%%word}
75167         x=posix/src/std
75168         echo ${x%%/*}
75169         posix

75170     ${parameter#word}
75171         x=$HOME/src/cmd
75172         echo ${x#$HOME}
75173         /src/cmd

75174     ${parameter###word}
75175         x=/one/two/three
75176         echo ${x##*/}
75177         three

```

The double-quoting of patterns is different depending on where the double-quotes are placed:

```

75179     "$ {x#*}"    The <asterisk> is a pattern character.
75180     ${x#"*" }   The literal <asterisk> is quoted and not special.

```

75181 2.6.3 Command Substitution

75182 Command substitution allows the output of a command to be substituted in place of the
75183 command name itself. Command substitution shall occur when the command is enclosed as
75184 follows:

```
75185     $( command )
```

75186 or (backquoted version):

```
75187     ` command `
```

75188 The shell shall expand the command substitution by executing *command* in a subshell
75189 environment (see [Section 2.12](#)) and replacing the command substitution (the text of *command*
75190 plus the enclosing "\$ () " or backquotes) with the standard output of the command, removing
75191 sequences of one or more <newline> characters at the end of the substitution. Embedded
75192 <newline> characters before the end of the output shall not be removed; however, they may be
75193 treated as field delimiters and eliminated during field splitting, depending on the value of *IFS*
75194 and quoting that is in effect. If the output contains any null bytes, the behavior is unspecified.

75195 Within the backquoted style of command substitution, <backslash> shall retain its literal
75196 meaning, except when followed by: '\$', '`', or <backslash>. The search for the matching

75197 backquote shall be satisfied by the first unquoted non-escaped backquote; during this search, if a
 75198 non-escaped backquote is encountered within a shell comment, a here-document, an embedded
 75199 command substitution of the $\$(command)$ form, or a quoted string, undefined results occur. A
 75200 single-quoted or double-quoted string that begins, but does not end, within the "``...``"
 75201 sequence produces undefined results.

75202 With the $\$(command)$ form, all characters following the open parenthesis to the matching closing
 75203 parenthesis constitute the *command*. Any valid shell script can be used for *command*, except a
 75204 script consisting solely of redirections which produces unspecified results.

75205 The results of command substitution shall not be processed for further tilde expansion,
 75206 parameter expansion, command substitution, or arithmetic expansion. If a command
 75207 substitution occurs inside double-quotes, field splitting and pathname expansion shall not be
 75208 performed on the results of the substitution.

75209 Command substitution can be nested. To specify nesting within the backquoted version, the
 75210 application shall precede the inner backquotes with `<backslash>` characters; for example:

```
75211 \ `command`
```

75212 The syntax of the shell command language has an ambiguity for expansions beginning with
 75213 "`$((`", which can introduce an arithmetic expansion or a command substitution that starts with
 75214 a subshell. Arithmetic expansion has precedence; that is, the shell shall first determine whether
 75215 it can parse the expansion as an arithmetic expansion and shall only parse the expansion as a
 75216 command substitution if it determines that it cannot parse the expansion as an arithmetic
 75217 expansion. The shell need not evaluate nested expansions when performing this determination.
 75218 If it encounters the end of input without already having determined that it cannot parse the
 75219 expansion as an arithmetic expansion, the shell shall treat the expansion as an incomplete
 75220 arithmetic expansion and report a syntax error. A conforming application shall ensure that it
 75221 separates the "`$((`" and "`' (`" into two tokens (that is, separate them with white space) in a
 75222 command substitution that starts with a subshell. For example, a command substitution
 75223 containing a single subshell could be written as:

```
75224 $( (command) )
```

75225 **2.6.4 Arithmetic Expansion**

75226 Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and
 75227 substituting its value. The format for arithmetic expansion shall be as follows:

```
75228 $( (expression) )
```

75229 The expression shall be treated as if it were in double-quotes, except that a double-quote inside
 75230 the expression is not treated specially. The shell shall expand all tokens in the expression for
 75231 parameter expansion, command substitution, and quote removal.

75232 Next, the shell shall treat this as an arithmetic expression and substitute the value of the
 75233 expression. The arithmetic expression shall be processed according to the rules given in [Section](#)
 75234 [1.1.2.1](#), with the following exceptions:

75235 Only signed long integer arithmetic is required.

75236 Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in
 75237 the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.

75238 The *sizeof()* operator and the prefix and postfix "`++`" and "`--`" operators are not required.

75239 Selection, iteration, and jump statements are not supported.

75240 All changes to variables in an arithmetic expression shall be in effect after the arithmetic
75241 expansion, as in the parameter expansion "`{x=value}`".

75242 If the shell variable *x* contains a value that forms a valid integer constant, optionally including a
75243 leading <plus-sign> or <hyphen-minus>, then the arithmetic expansions "`((x))`" and
75244 "`(($x))`" shall return the same value.

75245 As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell
75246 may use a signed integer type with a rank larger than the rank of **signed long**. The shell may
75247 use a real-floating type instead of **signed long** as long as it does not affect the results in cases
75248 where there is no overflow. If the expression is invalid, or the contents of a shell variable used in
75249 the expression are not recognized by the shell, the expansion fails and the shell shall write a
75250 diagnostic message to standard error indicating the failure.

75251 Examples

75252 A simple example using arithmetic expansion:

```
75253 # repeat a command 100 times
75254 x=100
75255 while [ $x -gt 0 ]
75256 do
75257     command
75258     x=$(( $x-1 ))
75259 done
```

75260 2.6.5 Field Splitting

75261 After parameter expansion (Section 2.6.2), command substitution (Section 2.6.3), and arithmetic
75262 expansion (Section 2.6.4), the shell shall scan the results of expansions and substitutions that did
75263 not occur in double-quotes for field splitting and multiple fields can result.

75264 The shell shall treat each character of the *IFS* as a delimiter and use the delimiters as field
75265 terminators to split the results of parameter expansion, command substitution, and arithmetic
75266 expansion into fields.

75267 1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of
75268 <space>, <tab>, or <newline> characters at the beginning or end of the input shall be
75269 ignored and any sequence of those characters within the input shall delimit a field. For
75270 example, the input:

```
75271 <newline><space><tab>foo<tab><tab>bar<space>
```

75272 yields two fields, **foo** and **bar**.

75273 2. If the value of *IFS* is null, no field splitting shall be performed.

75274 3. Otherwise, the following rules shall be applied in sequence. The term "*IFS* white space"
75275 is used to mean any sequence (zero or more instances) of white-space characters that are
75276 in the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of
75277 <space> and <tab> characters is considered *IFS* white space).

- 75278 a. *IFS* white space shall be ignored at the beginning and end of the input.
- 75279 b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along
75280 with any adjacent *IFS* white space, shall delimit a field, as described previously.
- 75281 c. Non-zero-length *IFS* white space shall delimit a field.

75282 2.6.6 Pathname Expansion

75283 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be
75284 expanded using the algorithm described in [Section 2.13](#), qualified by the rules in [Section 2.13.3](#).

75285 2.6.7 Quote Removal

75286 The quote characters (<backslash>, single-quote, and double-quote) that were present in the
75287 original word shall be removed unless they have themselves been quoted.

75288 2.7 Redirection

75289 Redirection is used to open and close files for the current shell execution environment (see
75290 [Section 2.12](#)) or for any command. Redirection operators can be used with numbers representing
75291 file descriptors (see [XBD Section 3.166](#)) as described below.

75292 The overall format used for redirection is:

```
75293 [n]redir-op word
```

75294 The number *n* is an optional decimal number designating the file descriptor number; the
75295 application shall ensure it is delimited from any preceding text and immediately precede the
75296 redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the
75297 redirection expression. For example:

```
75298 echo \2>a
```

75299 writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is
75300 recognized. For example:

```
75301 echo 2\>a
```

75302 writes the characters 2>a to standard output. The optional number, redirection operator, and
75303 *word* shall not appear in the arguments provided to the command to be executed (if any).

75304 Open files are represented by decimal numbers starting with zero. The largest possible value is
75305 implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for
75306 use by the application. These numbers are called “file descriptors”. The values 0, 1, and 2 have
75307 special meaning and conventional uses and are implied by certain redirection operations; they
75308 are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs
75309 usually take their input from standard input, and write output on standard output. Error
75310 messages are usually written on standard error. The redirection operators can be preceded by
75311 one or more digits (with no intervening <blank> characters allowed) to designate the file
75312 descriptor number.

75313 If the redirection operator is "<<" or "<<-", the word that follows the redirection operator shall
75314 be subjected to quote removal; it is unspecified whether any of the other expansions occur. For
75315 the other redirection operators, the word that follows the redirection operator shall be subjected

75316 to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and
 75317 quote removal. Pathname expansion shall not be performed on the word by a non-interactive
 75318 shell; an interactive shell may perform it, but shall do so only when the expansion would result
 75319 in one word.

75320 If more than one redirection operator is specified with a command, the order of evaluation is
 75321 from beginning to end.

75322 A failure to open or create a file shall cause a redirection to fail.

75323 2.7.1 Redirecting Input

75324 Input redirection shall cause the file whose name results from the expansion of *word* to be
 75325 opened for reading on the designated file descriptor, or standard input if the file descriptor is
 75326 not specified.

75327 The general format for redirecting input is:

75328 `[n]<word`

75329 where the optional *n* represents the file descriptor number. If the number is omitted, the
 75330 redirection shall refer to standard input (file descriptor 0).

75331 2.7.2 Redirecting Output

75332 The two general formats for redirecting output are:

75333 `[n]>word`

75334 `[n]>|word`

75335 where the optional *n* represents the file descriptor number. If the number is omitted, the
 75336 redirection shall refer to standard output (file descriptor 1).

75337 Output redirection using the '>' format shall fail if the *noclobber* option is set (see the
 75338 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.
 75339 Otherwise, redirection using the '>' or '>|' formats shall cause the file whose name results
 75340 from the expansion of *word* to be created and opened for output on the designated file
 75341 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;
 75342 otherwise, it shall be truncated to be an empty file after being opened.

75343 2.7.3 Appending Redirected Output

75344 Appended output redirection shall cause the file whose name results from the expansion of
 75345 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*
 75346 function as defined in the System Interfaces volume of POSIX.1-2017 was called with the
 75347 `O_APPEND` flag. If the file does not exist, it shall be created.

75348 The general format for appending redirected output is as follows:

75349 `[n]>>word`

75350 where the optional *n* represents the file descriptor number. If the number is omitted, the
 75351 redirection refers to standard output (file descriptor 1).

75352 2.7.4 Here-Document

75353 The redirection operators "`<<`" and "`<<-`" both allow redirection of subsequent lines read by
75354 the shell to the input of a command. The redirected lines are known as a "here-document".

75355 The here-document shall be treated as a single word that begins after the next `<newline>` and
75356 continues until there is a line containing only the delimiter and a `<newline>`, with no `<blank>`
75357 characters in between. Then the next here-document starts, if there is one. The format is as
75358 follows:

```
75359 [n]<<word
75360     here-document
75361     delimiter
```

75362 where the optional *n* represents the file descriptor number. If the number is omitted, the here-
75363 document refers to standard input (file descriptor 0). It is unspecified whether the file descriptor
75364 is opened as a regular file, a special file, or a pipe. Portable applications cannot rely on the file
75365 descriptor being seekable (see XSH [lseek\(\)](#)).

75366 If any part of *word* is quoted, the delimiter shall be formed by performing quote removal on
75367 *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the
75368 *word* itself.

75369 If no part of *word* is quoted, all lines of the here-document shall be expanded for parameter
75370 expansion, command substitution, and arithmetic expansion. In this case, the `<backslash>` in the
75371 input behaves as the `<backslash>` inside double-quotes (see [Section 2.2.3](#)). However, the double-
75372 quote character (`' '`) shall not be treated specially within a here-document, except when the
75373 double-quote appears within "`$ ()`", "`$ ` ``", or "`${ }`".

75374 If the redirection operator is "`<<-`", all leading `<tab>` characters shall be stripped from input
75375 lines and the line containing the trailing delimiter. If more than one "`<<`" or "`<<-`" operator is
75376 specified on a line, the here-document associated with the first operator shall be supplied first
75377 by the application and shall be read first by the shell.

75378 When a here-document is read from a terminal device and the shell is interactive, it shall write
75379 the contents of the variable *PS2*, processed as described in [Section 2.5.3](#), to standard error before
75380 reading each line of input until the delimiter has been recognized.

75381 Examples

75382 An example of a here-document follows:

```
75383 cat <<eof1; cat <<eof2
75384 Hi,
75385 eof1
75386 Helene.
75387 eof2
```

75388 2.7.5 Duplicating an Input File Descriptor

75389 The redirection operator:

75390 `[n]<&word`

75391 shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one
75392 or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be
75393 made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a
75394 file descriptor already open for input, a redirection error shall result; see [Section 2.8.1](#). If *word*
75395 evaluates to '-', file descriptor *n*, or standard input if *n* is not specified, shall be closed.
75396 Attempts to close a file descriptor that is not open shall not constitute an error. If *word* evaluates
75397 to something else, the behavior is unspecified.

75398 2.7.6 Duplicating an Output File Descriptor

75399 The redirection operator:

75400 `[n]>&word`

75401 shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to
75402 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall
75403 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent
75404 a file descriptor already open for output, a redirection error shall result; see [Section 2.8.1](#). If *word*
75405 evaluates to '-', file descriptor *n*, or standard output if *n* is not specified, is closed. Attempts to
75406 close a file descriptor that is not open shall not constitute an error. If *word* evaluates to something
75407 else, the behavior is unspecified.

75408 2.7.7 Open File Descriptors for Reading and Writing

75409 The redirection operator:

75410 `[n]<>word`

75411 shall cause the file whose name is the expansion of *word* to be opened for both reading and
75412 writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does
75413 not exist, it shall be created.

75414 2.8 Exit Status and Errors

75415 2.8.1 Consequences of Shell Errors

75416 Certain errors shall cause the shell to write a diagnostic message to standard error and exit as
75417 shown in the following table:

Error	Non-Interactive Shell	Interactive Shell	Shell Diagnostic Message Required
Shell language syntax error	shall exit	shall not exit	yes
Special built-in utility error	shall exit	shall not exit	no ¹
Other utility (not a special built-in) error	shall not exit	shall not exit	no ²
Redirection error with special built-in utilities	shall exit	shall not exit	yes
Redirection error with compound commands	may exit ³	shall not exit	yes
Redirection error with function execution	may exit ³	shall not exit	yes
Redirection error with other utilities (not special built-ins)	shall not exit	shall not exit	yes
Variable assignment error	shall exit	shall not exit	yes
Expansion error	shall exit	shall not exit	yes
Command not found	may exit	shall not exit	yes

Notes:

1. Although special built-ins are part of the shell, a diagnostic message written by a special built-in is not considered to be a shell diagnostic message, and can be redirected like any other utility.
2. The shell is not required to write a diagnostic message, but the utility itself shall write a diagnostic message if required to do so.
3. A future version of this standard may require the shell to not exit in this condition.

An expansion error is one that occurs when the shell expansions define in [Section 2.6](#) are carried out (for example, "`{x!y}`", because `!` is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as "shall exit" or "may exit" occur in a subshell environment, the shell shall (respectively, may) exit from the subshell environment with a non-zero status and continue in the environment from which that subshell environment was invoked.

In all of the cases shown in the table where an interactive shell is required not to exit, the shell shall not perform any further processing of the command in which the error occurred.

2.8.2 Exit Status for Commands

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities is documented in this section. The exit status of the standard utilities is documented in their respective sections.

If a command is not found, the exit status shall be 127. If the command name is found, but it is not an executable utility, the exit status shall be 126. Applications that invoke utilities without using the shell should use these exit status values to report similar errors.

If a command fails during word expansion or redirection, its exit status shall be between 1 and 125 inclusive.

Internally, for purposes of deciding whether a command exits with a non-zero exit status, the

75461 shell shall recognize the entire status value retrieved for the command by the equivalent of the
 75462 *wait()* function WEXITSTATUS macro (as defined in the System Interfaces volume of
 75463 POSIX.1-2017). When reporting the exit status with the special parameter '?', the shell shall
 75464 report the full eight bits of exit status available. The exit status of a command that terminated
 75465 because it received a signal shall be reported as greater than 128.

75466 2.9 Shell Commands

75467 This section describes the basic structure of shell commands. The following command
 75468 descriptions each describe a format of the command that is only used to aid the reader in
 75469 recognizing the command type, and does not formally represent the syntax. In particular, the
 75470 representations include spacing between tokens in some places where <blank>s would not be
 75471 necessary (when one of the tokens is an operator). Each description discusses the semantics of
 75472 the command; for a formal definition of the command language, consult [Section 2.10](#).

75473 A *command* is one of the following:

- 75474 Simple command (see [Section 2.9.1](#))
- 75475 Pipeline (see [Section 2.9.2](#))
- 75476 List compound-list (see [Section 2.9.3](#))
- 75477 Compound command (see [Section 2.9.4](#))
- 75478 Function definition (see [Section 2.9.5](#))

75479 Unless otherwise stated, the exit status of a command shall be that of the last simple command
 75480 executed by the command. There shall be no limit on the size of any shell command other than
 75481 that imposed by the underlying system (memory constraints, {ARG_MAX}, and so on).

75482 2.9.1 Simple Commands

75483 A “simple command” is a sequence of optional variable assignments and redirections, in any
 75484 sequence, optionally followed by words and redirections, terminated by a control operator.

75485 When a given simple command is required to be executed (that is, when any conditional
 75486 construct such as an AND-OR list or a **case** statement has not bypassed the simple command),
 75487 the following expansions, assignments, and redirections shall all be performed from the
 75488 beginning of the command text to the end:

- 75489 1. The words that are recognized as variable assignments or redirections according to
 75490 [Section 2.10.2](#) are saved for processing in steps 3 and 4.
- 75491 2. The words that are not variable assignments or redirections shall be expanded. If any
 75492 fields remain following their expansion, the first field shall be considered the command
 75493 name and remaining fields are the arguments for the command.
- 75494 3. Redirections shall be performed as described in [Section 2.7](#).
- 75495 4. Each variable assignment shall be expanded for tilde expansion, parameter expansion,
 75496 command substitution, arithmetic expansion, and quote removal prior to assigning the
 75497 value.

75498 In the preceding list, the order of steps 3 and 4 may be reversed if no command name results
 75499 from step 2 or if the command name matches the name of a special built-in utility; see [Section](#)
 75500 [2.14](#).

75501 Variable assignments shall be performed as follows:

75502 If no command name results, variable assignments shall affect the current execution
75503 environment.

75504 If the command name is not a special built-in utility or function, the variable assignments
75505 shall be exported for the execution environment of the command and shall not affect the
75506 current execution environment except as a side-effect of the expansions performed in step
75507 4. In this case it is unspecified:

75508 ‡ whether or not the assignments are visible for subsequent expansions in step 4

75509 ‡ whether variable assignments made as side-effects of these expansions are visible for
75510 subsequent expansions in step 4, or in the current shell execution environment, or
75511 both

75512 If the command name is a standard utility implemented as a function (see XBD [Section](#)
75513 [4.22](#)), the effect of variable assignments shall be as if the utility was not implemented as a
75514 function.

75515 If the command name is a special built-in utility, variable assignments shall affect the
75516 current execution environment. Unless the *set -a* option is on (see *set*), it is unspecified:

75517 ‡ whether or not the variables gain the *export* attribute during the execution of the
75518 special built-in utility

75519 ‡ whether or not *export* attributes gained as a result of the variable assignments persist
75520 after the completion of the special built-in utility

75521 If the command name is a function that is not a standard utility implemented as a function,
75522 variable assignments shall affect the current execution environment during the execution
75523 of the function. It is unspecified:

75524 ‡ whether or not the variable assignments persist after the completion of the function

75525 ‡ whether or not the variables gain the *export* attribute during the execution of the
75526 function

75527 ‡ whether or not *export* attributes gained as a result of the variable assignments persist
75528 after the completion of the function (if variable assignments persist after the
75529 completion of the function)

75530 If any of the variable assignments attempt to assign a value to a variable for which the *readonly*
75531 attribute is set in the current shell environment (regardless of whether the assignment is made in
75532 that environment), a variable assignment error shall occur. See [Section 2.8.1](#) for the consequences
75533 of these errors.

75534 If there is no command name, any redirections shall be performed in a subshell environment; it
75535 is unspecified whether this subshell environment is the same one as that used for a command
75536 substitution within the command. (To affect the current execution environment, see the *exec*
75537 special built-in.) If any of the redirections performed in the current shell execution environment
75538 fail, the command shall immediately fail with an exit status greater than zero, and the shell shall
75539 write an error message indicating the failure. See [Section 2.8.1](#) for the consequences of these
75540 failures on interactive and non-interactive shells.

75541 If there is a command name, execution shall continue as described in [Section 2.9.1.1](#). If there is
75542 no command name, but the command contained a command substitution, the command shall
75543 complete with the exit status of the last command substitution performed. Otherwise, the
75544 command shall complete with a zero exit status.

75545 2.9.1.1 Command Search and Execution

75546 If a simple command results in a command name and an optional list of arguments, the
75547 following actions shall be performed:

75548 1. If the command name does not contain any <slash> characters, the first successful step in
75549 the following sequence shall occur:

75550 a. If the command name matches the name of a special built-in utility, that special
75551 built-in utility shall be invoked.

75552 b. If the command name matches the name of a utility listed in the following table,
75553 the results are unspecified.

75554	<i>alloc</i>	<i>comparguments</i>	<i>comptry</i>	<i>history</i>	<i>pushd</i>
75555	<i>autoload</i>	<i>compcall</i>	<i>compvalues</i>	<i>hist</i>	<i>readarray</i>
75556	<i>bind</i>	<i>compctl</i>	<i>declare</i>	<i>let</i>	<i>repeat</i>
75557	<i>bindkey</i>	<i>compdescribe</i>	<i>dirs</i>	<i>local</i>	<i>savehistory</i>
75558	<i>builtin</i>	<i>compfiles</i>	<i>disable</i>	<i>login</i>	<i>source</i>
75559	<i>bye</i>	<i>compgen</i>	<i>disown</i>	<i>logout</i>	<i>shopt</i>
75560	<i>caller</i>	<i>compgroups</i>	<i>dosh</i>	<i>map</i>	<i>stop</i>
75561	<i>cap</i>	<i>complete</i>	<i>echoct</i>	<i>mapfile</i>	<i>suspend</i>
75562	<i>chdir</i>	<i>compquote</i>	<i>echoit</i>	<i>popd</i>	<i>typeset</i>
75563	<i>clone</i>	<i>comptags</i>	<i>help</i>	<i>print</i>	<i>whence</i>

75564 c. If the command name matches the name of a function known to this shell, the
75565 function shall be invoked as described in [Section 2.9.5](#). If the implementation has
75566 provided a standard utility in the form of a function, it shall not be recognized at
75567 this point. It shall be invoked in conjunction with the path search in step 1e.

75568 XSI d. If the command name matches the name of the *type* or *ulimit* utility, or
75569 of a utility listed in the following table, that utility shall be invoked.

75570	<i>alias</i>	<i>false</i>	<i>hash</i>	<i>pwd</i>	<i>unalias</i>
75571	<i>bg</i>	<i>fc</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
75572	<i>cd</i>	<i>fg</i>	<i>kill</i>	<i>true</i>	
75573	<i>command</i>	<i>getopts</i>	<i>newgrp</i>	<i>umask</i>	

75574 e. Otherwise, the command shall be searched for using the *PATH* environment
75575 variable as described in XBD [Chapter 8](#):

75576 i. If the search is successful:

75577 a. If the system has implemented the utility as a regular built-in or as a
75578 shell function, it shall be invoked at this point in the path search.

75579 b. Otherwise, the shell executes the utility in a separate utility
75580 environment (see [Section 2.12](#)) with actions equivalent to calling the
75581 *execl()* function as defined in the System Interfaces volume of
75582 POSIX.1-2017 with the *path* argument set to the pathname resulting
75583 from the search, *arg0* set to the command name, and the remaining
75584 *execl()* arguments set to the command arguments (if any) and the
75585 null terminator.

75586 If the *execl()* function fails due to an error equivalent to the
75587 [ENOEXEC] error defined in the System Interfaces volume of
75588 POSIX.1-2017, the shell shall execute a command equivalent to
75589 having a shell invoked with the pathname resulting from the search
75590 as its first operand, with any remaining arguments passed to the new

75591 shell, except that the value of "\$0" in the new shell may be set to the
 75592 command name. If the executable file is not a text file, the shell may
 75593 bypass this command execution. In this case, it shall write an error
 75594 message, and shall return an exit status of 126.

75595 It is unspecified whether environment variables that were passed to
 75596 the shell when it was invoked, but were not used to initialize shell
 75597 variables (see Section 2.5.3) because they had invalid names, are
 75598 included in the environment passed to *execl()* and (if *execl()* fails as
 75599 described above) to the new shell.

75600 Once a utility has been searched for and found (either as a result of this
 75601 specific search or as part of an unspecified shell start-up activity), an
 75602 implementation may remember its location and need not search for the
 75603 utility again unless the *PATH* variable has been the subject of an assignment.
 75604 If the remembered location fails for a subsequent invocation, the shell shall
 75605 repeat the search to find the new location for the utility, if any.

75606 ii. If the search is unsuccessful, the command shall fail with an exit status of
 75607 127 and the shell shall write an error message.

75608 2. If the command name contains at least one <slash>, the shell shall execute the utility in a
 75609 separate utility environment with actions equivalent to calling the *execl()* function
 75610 defined in the System Interfaces volume of POSIX.1-2017 with the *path* and *arg0*
 75611 arguments set to the command name, and the remaining *execl()* arguments set to the
 75612 command arguments (if any) and the null terminator.

75613 If the *execl()* function fails due to an error equivalent to the [ENOEXEC] error, the shell
 75614 shall execute a command equivalent to having a shell invoked with the command name
 75615 as its first operand, with any remaining arguments passed to the new shell. If the
 75616 executable file is not a text file, the shell may bypass this command execution. In this case,
 75617 it shall write an error message and shall return an exit status of 126.

75618 It is unspecified whether environment variables that were passed to the shell when it was
 75619 invoked, but were not used to initialize shell variables (see Section 2.5.3) because they had
 75620 invalid names, are included in the environment passed to *execl()* and (if *execl()* fails as
 75621 described above) to the new shell.

75622 If the utility would be executed with file descriptor 0, 1, or 2 closed, implementations may
 75623 execute the utility with the file descriptor open to an unspecified file. If a standard utility or a
 75624 conforming application is executed with file descriptor 0 not open for reading or with file
 75625 descriptor 1 or 2 not open for writing, the environment in which the utility or application is
 75626 executed shall be deemed non-conforming, and consequently the utility or application might not
 75627 behave as described in this standard.

75628 2.9.2 Pipelines

75629 A *pipeline* is a sequence of one or more commands separated by the control operator '|'. For
 75630 each command but the last, the shell shall connect the standard output of the command to the
 75631 standard input of the next command as if by creating a pipe and passing the write end of the
 75632 pipe as the standard output of the command and the read end of the pipe as the standard input
 75633 of the next command.

75634 The format for a pipeline is:

```
75635 [!] command1 [ | command2 ... ]
```

75636 If the pipeline begins with the reserved word `!` and *command1* is a subshell command, the
 75637 application shall ensure that the `(` operator at the beginning of *command1* is separated from the `!`
 75638 by one or more `<blank>` characters. The behavior of the reserved word `!` immediately followed
 75639 by the `(` operator is unspecified.

75640 The standard output of *command1* shall be connected to the standard input of *command2*. The
 75641 standard input, standard output, or both of a command shall be considered to be assigned by
 75642 the pipeline before any redirection specified by redirection operators that are part of the
 75643 command (see [Section 2.7](#)).

75644 If the pipeline is not in the background (see [Section 2.9.3.1](#)), the shell shall wait for the last
 75645 command specified in the pipeline to complete, and may also wait for all commands to
 75646 complete.

75647 **Exit Status**

75648 If the pipeline does not begin with the `!` reserved word, the exit status shall be the exit status of
 75649 the last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of
 75650 the exit status of the last command. That is, if the last command returns zero, the exit status shall
 75651 be 1; if the last command returns greater than zero, the exit status shall be zero.

75652 **2.9.3 Lists**

75653 An *AND-OR list* is a sequence of one or more pipelines separated by the operators `&&` and
 75654 `||`.

75655 A *list* is a sequence of one or more AND-OR lists separated by the operators `;` and `'&'`.

75656 The operators `&&` and `||` shall have equal precedence and shall be evaluated with left
 75657 associativity. For example, both of the following commands write solely **bar** to standard output:

```
75658 false && echo foo || echo bar
75659 true || echo foo && echo bar
```

75660 A `;` separator or a `'&'` or `<newline>` terminator shall cause the preceding AND-OR list to be
 75661 executed sequentially; an `'&'` separator or terminator shall cause asynchronous execution of the
 75662 preceding AND-OR list.

75663 The term “compound-list” is derived from the grammar in [Section 2.10](#); it is equivalent to a
 75664 sequence of *lists*, separated by `<newline>` characters, that can be preceded or followed by an
 75665 arbitrary number of `<newline>` characters.

75666 **Examples**

75667 The following is an example that illustrates `<newline>` characters in compound-lists:

```
75668 while
75669     # a couple of <newline>s
75670     # a list
75671     date && who || ls; cat file
75672     # a couple of <newline>s
75673     # another list
75674     wc file > output & true
75675 do
75676     # 2 lists
```



```

75677         ls
75678         cat file
75679         done

```

75680 2.9.3.1 Asynchronous Lists

75681 If a command is terminated by the control operator <ampersand> ('&'), the shell shall execute
 75682 the command asynchronously in a subshell. This means that the shell shall not wait for the
 75683 command to finish before executing the next command.

75684 The format for running a command in the background is:

```
75685 command1 & [command2 & ... ]
```

75686 If job control is disabled (see *set*, **-m**), the standard input for an asynchronous list, before any
 75687 explicit redirections are performed, shall be considered to be assigned to a file that has the same
 75688 properties as **/dev/null**. This shall not happen if job control is enabled. In all cases, explicit
 75689 redirection of standard input shall override this activity.

75690 When an element of an asynchronous list (the portion of the list ended by an <ampersand>,
 75691 such as *command1*, above) is started by the shell, the process ID of the last command in the
 75692 asynchronous list element shall become known in the current shell execution environment; see
 75693 [Section 2.12](#). This process ID shall remain known until:

- 75694 1. The command terminates and the application waits for the process ID.
- 75695 2. Another asynchronous list is invoked before "\$!" (corresponding to the previous
 75696 asynchronous list) is expanded in the current execution environment.

75697 The implementation need not retain more than the {CHILD_MAX} most recent entries in its list
 75698 of known process IDs in the current shell execution environment.

75699 **Exit Status**

75700 The exit status of an asynchronous list shall be zero.

75701 2.9.3.2 Sequential Lists

75702 Commands that are separated by a <semicolon> (';') shall be executed sequentially.

75703 The format for executing commands sequentially shall be:

```
75704 command1 [; command2] ...
```

75705 Each command shall be expanded and executed in the order specified.

75706 **Exit Status**

75707 The exit status of a sequential list shall be the exit status of the last command in the list.

75708 2.9.3.3 AND Lists

75709 The control operator "&&" denotes an AND list. The format shall be:

```
75710 command1 [ && command2] ...
```

75711 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,
 75712 until a command has a non-zero exit status or there are no more commands left to execute. The

75713 commands are expanded only if they are executed.

75714 **Exit Status**

75715 The exit status of an AND list shall be the exit status of the last command that is executed in the
75716 list.

75717 2.9.3.4 *OR Lists*

75718 The control operator "`||`" denotes an OR List. The format shall be:

75719 `command1 [|| command2] . . .`

75720 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and
75721 so on, until a command has a zero exit status or there are no more commands left to execute.

75722 **Exit Status**

75723 The exit status of an OR list shall be the exit status of the last command that is executed in the
75724 list.

75725 **2.9.4 Compound Commands**

75726 The shell has several programming constructs that are “compound commands”, which provide
75727 control flow for commands. Each of these compound commands has a reserved word or control
75728 operator at the beginning, and a corresponding terminator reserved word or operator at the end.
75729 In addition, each can be followed by redirections on the same line as the terminator. Each
75730 redirection shall apply to all the commands within the compound command that do not
75731 explicitly override that redirection.

75732 2.9.4.1 *Grouping Commands*

75733 The format for grouping commands is as follows:

75734 (`compound-list`) Execute *compound-list* in a subshell environment; see [Section 2.12](#).
75735 Variable assignments and built-in commands that affect the environment
75736 shall not remain in effect after the list finishes.

75737 If a character sequence beginning with "`((`" would be parsed by the shell
75738 as an arithmetic expansion if preceded by a '`$`', shells which implement
75739 an extension whereby "`((expression))`" is evaluated as an arithmetic
75740 expression may treat the "`((`" as introducing as an arithmetic evaluation
75741 instead of a grouping command. A conforming application shall ensure
75742 that it separates the two leading '`(`' characters with white space to
75743 prevent the shell from performing an arithmetic evaluation.

75744 { `compound-list` ; } Execute *compound-list* in the current process environment. The semicolon
75745 shown here is an example of a control operator delimiting the } reserved
75746 word. Other delimiters are possible, as shown in [Section 2.10](#); a
75747 <newline> is frequently used.

75748 **Exit Status**75749 The exit status of a grouping command shall be the exit status of *compound-list*.75750 2.9.4.2 *The for Loop*75751 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**
75752 loop requires that the reserved words **do** and **done** be used to delimit the sequence of
75753 commands.75754 The format for the **for** loop is as follows:75755 `for name [in [word ...]]`
75756 `do`
75757 `compound-list`
75758 `done`75759 First, the list of words following **in** shall be expanded to generate a list of items. Then, the
75760 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no
75761 items result from the expansion, the *compound-list* shall not be executed. Omitting:75762 `in word...`

75763 shall be equivalent to:

75764 `in "$@"`75765 **Exit Status**75766 The exit status of a **for** command shall be the exit status of the last command that executes. If
75767 there are no items, the exit status shall be zero.75768 2.9.4.3 *Case Conditional Construct*75769 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of
75770 several *patterns* (see [Section 2.13](#)) that is matched by the string resulting from the tilde
75771 expansion, parameter expansion, command substitution, arithmetic expansion, and quote
75772 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to
75773 be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|'
75774 symbol. The control operator ')' terminates a list of patterns corresponding to a given action.
75775 The *compound-list* for each list of patterns, with the possible exception of the last, shall be
75776 terminated with ";;". The **case** construct terminates with the reserved word **esac** (**case**
75777 reversed).75778 The format for the **case** construct is as follows:75779 `case word in`
75780 `[(pattern1) compound-list ;;`
75781 `[[(pattern[| pattern] ...) compound-list ;;] ...`
75782 `[[(pattern[| pattern] ...) compound-list]`
75783 `esac`75784 The ";;" is optional for the last *compound-list*.75785 In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-*
75786 *list* shall be subjected to tilde expansion, parameter expansion, command substitution, and
75787 arithmetic expansion, and the result of these expansions shall be compared against the
75788 expansion of *word*, according to the rules described in [Section 2.13](#) (which also describes the

75789 effect of quoting parts of the pattern). After the first match, no more patterns shall be expanded,
 75790 and the *compound-list* shall be executed. The order of expansion and comparison of multiple
 75791 *patterns* that label a *compound-list* statement is unspecified.

75792 **Exit Status**

75793 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be
 75794 the exit status of the last command executed in the *compound-list*.

75795 2.9.4.4 *The if Conditional Construct*

75796 The **if** command shall execute a *compound-list* and use its exit status to determine whether to
 75797 execute another *compound-list*.

75798 The format for the **if** construct is as follows:

```
75799 if compound-list
75800 then
75801     compound-list
75802 [elif compound-list
75803 then
75804     compound-list] ...
75805 [else
75806     compound-list]
75807 fi
```

75808 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be
 75809 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,
 75810 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command
 75811 shall complete. Otherwise, the **else** *compound-list* shall be executed.

75812 **Exit Status**

75813 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that
 75814 was executed, or zero, if none was executed.

75815 2.9.4.5 *The while Loop*

75816 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has
 75817 a zero exit status.

75818 The format of the **while** loop is as follows:

```
75819 while compound-list-1
75820 do
75821     compound-list-2
75822 done
```

75823 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command
 75824 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

75825 **Exit Status**

75826 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or
75827 zero if none was executed.

75828 2.9.4.6 *The until Loop*

75829 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has
75830 a non-zero exit status.

75831 The format of the **until** loop is as follows:

```
75832 until compound-list-1
75833 do
75834     compound-list-2
75835 done
```

75836 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command
75837 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

75838 **Exit Status**

75839 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or
75840 zero if none was executed.

75841 **2.9.5 Function Definition Command**

75842 A function is a user-defined name that is used as a simple command to call a compound
75843 command with new positional parameters. A function is defined with a “function definition
75844 command”.

75845 The format of a function definition command is as follows:

```
75846 fname ( ) compound-command [io-redirect ...]
```

75847 The function is named *fname*; the application shall ensure that it is a name (see XBD [Section](#)
75848 [3.235](#)) and that it is not the name of a special built-in utility. An implementation may allow other
75849 characters in a function name as an extension. The implementation shall maintain separate name
75850 spaces for functions and variables.

75851 The argument *compound-command* represents a compound command, as described in [Section](#)
75852 [2.9.4](#).

75853 When the function is declared, none of the expansions in [Section 2.6](#) shall be performed on the
75854 text in *compound-command* or *io-redirect*; all expansions shall be performed as normal each time
75855 the function is called. Similarly, the optional *io-redirect* redirections and any variable assignments
75856 within *compound-command* shall be performed during the execution of the function itself, not the
75857 function definition. See [Section 2.8.1](#) for the consequences of failures of these operations on
75858 interactive and non-interactive shells.

75859 When a function is executed, it shall have the syntax-error properties described for special built-
75860 in utilities in the first item in the enumerated list at the beginning of [Section 2.14](#).

75861 The *compound-command* shall be executed whenever the function name is specified as the name
75862 of a simple command (see [Section 2.9.1.1](#)). The operands to the command temporarily shall
75863 become the positional parameters during the execution of the *compound-command*; the special
75864 parameter '# ' also shall be changed to reflect the number of operands. The special parameter 0
75865 shall be unchanged. When the function completes, the values of the positional parameters and

75866 the special parameter '#' shall be restored to the values they had before the function was
 75867 executed. If the special built-in *return* (see *return*) is executed in the *compound-command*, the
 75868 function completes and execution shall resume with the next command after the function call.

75869 **Exit Status**

75870 The exit status of a function definition shall be zero if the function was declared successfully;
 75871 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit
 75872 status of the last command executed by the function.

75873 **2.10 Shell Grammar**

75874 The following grammar defines the Shell Command Language. This formal syntax shall take
 75875 precedence over the preceding text syntax description.

75876 **2.10.1 Shell Grammar Lexical Conventions**

75877 The input language to the shell must be first recognized at the character level. The resulting
 75878 tokens shall be classified by their immediate context according to the following rules (applied in
 75879 order). These rules shall be used to determine what a "token" is that is subject to parsing at the
 75880 token level. The rules for token recognition in [Section 2.3](#) shall apply.

- 75881 1. If the token is an operator, the token identifier for that operator shall result.
- 75882 2. If the string consists solely of digits and the delimiter character is one of '<' or '>', the
 75883 token identifier **IO_NUMBER** shall be returned.
- 75884 3. Otherwise, the token identifier **TOKEN** results.

75885 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields
 75886 **WORD**, a **NAME**, an **ASSIGNMENT_WORD**, or one of the reserved words below, dependent
 75887 upon the context. Some of the productions in the grammar below are annotated with a rule
 75888 number from the following list. When a **TOKEN** is seen where one of those annotated
 75889 productions could be used to reduce the symbol, the applicable rule shall be applied to convert
 75890 the token identifier type of the **TOKEN** to a token identifier acceptable at that point in the
 75891 grammar. The reduction shall then proceed based upon the token identifier type yielded by the
 75892 rule applied. When more than one rule applies, the highest numbered rule shall apply (which in
 75893 turn may refer to another rule). (Note that except in rule 7, the presence of an '=' in the token
 75894 has no effect.)

75895 The **WORD** tokens shall have the word expansion rules applied to them immediately before the
 75896 associated command is executed, not at the time the command is parsed.

75897 **2.10.2 Shell Grammar Rules**

- 75898 1. [Command Name]

75899 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word
 75900 shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any
 75901 state where only a reserved word could be the next correct token, proceed as above.

75902 **Note:** Because at this point <quotation-mark> characters are retained in the token, quoted
 75903 strings cannot be recognized as reserved words. This rule also implies that reserved
 75904 words are not recognized except in certain positions in the input, such as after a

75905 <newline> or <semicolon>; the grammar presumes that if the reserved word is
 75906 intended, it is properly delimited by the user, and does not attempt to reflect that
 75907 requirement directly. Also note that line joining is done before tokenization, as described
 75908 in [Section 2.2.1](#), so escaped <newline> characters are already removed at this point.

75909 Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or
 75910 applies globally.

75911 2. [Redirection to or from filename]

75912 The expansions specified in [Section 2.7](#) shall occur. As specified there, exactly one field
 75913 can result (or the result is unspecified), and there are additional requirements on
 75914 pathname expansion.

75915 3. [Redirection from here-document]

75916 Quote removal shall be applied to the word to determine the delimiter that is used to find
 75917 the end of the here-document that begins after the next <newline>.

75918 4. [Case statement termination]

75919 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall
 75920 result. Otherwise, the token **WORD** shall be returned.

75921 5. [NAME in for]

75922 When the **TOKEN** meets the requirements for a name (see XBD [Section 3.235](#)), the token
 75923 identifier **NAME** shall result. Otherwise, the token **WORD** shall be returned.

75924 6. [Third word of for and case]

75925 a. [**case** only]

75926 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall
 75927 result. Otherwise, the token **WORD** shall be returned.

75928 b. [**for** only]

75929 When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in**
 75930 or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.

75931 (For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If
 75932 <newline> characters are present at the indicated location, it is the token after them that is
 75933 treated in this fashion.)

75934 7. [Assignment preceding command name]

75935 a. [When the first word]

75936 If the **TOKEN** does not contain the character '=' , rule 1 is applied. Otherwise, 7b
 75937 shall be applied.

75938 b. [Not the first word]

75939 If the **TOKEN** contains an unquoted (as determined while applying rule 4 from
 75940 [Section 2.3](#)) <equals-sign> character that is not part of an embedded parameter
 75941 expansion, command substitution, or arithmetic expansion construct (as
 75942 determined while applying rule 5 from [Section 2.3](#)):

75943 **if** the **TOKEN** begins with '=' , then rule 1 shall be applied.

75944 **if** all the characters in the **TOKEN** preceding the first such <equals-sign>
 75945 form a valid name (see XBD [Section 3.235](#), on page 71), the token
 75946 **ASSIGNMENT_WORD** shall be returned.

75947 ‡ Otherwise, it is unspecified whether rule 1 is applied or
 75948 **ASSIGNMENT_WORD** is returned.

75949 Otherwise, rule 1 shall be applied.

75950 Assignment to the name within a returned **ASSIGNMENT_WORD** token shall occur as
 75951 specified in [Section 2.9.1](#).

75952 8. [NAME in function]

75953 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word
 75954 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token
 75955 identifier **NAME** shall result. Otherwise, rule 7 applies.

75956 9. [Body of function]

75957 Word expansion and assignment shall never occur, even when required by the rules
 75958 above, when this rule is being parsed. Each **TOKEN** that might either be expanded or
 75959 have assignment applied to it shall instead be returned as a single **WORD** consisting only
 75960 of characters that are exactly the token described in [Section 2.3](#).

```

75961 /* -----
75962 The grammar symbols
75963 ----- */
75964 %token WORD
75965 %token ASSIGNMENT_WORD
75966 %token NAME
75967 %token NEWLINE
75968 %token IO_NUMBER

75969 /* The following are the operators (see XBD Section 3.260)
75970 containing more than one character. */

75971 %token AND_IF OR_IF DSEMI
75972 /* '&&' '||' ';;' */

75973 %token DLESS DGREAT LESSAND GREATAND LESSGREAT DLESSDASH
75974 /* '<<' '>>' '<&' '>&' '<>' '<<-' */

75975 %token CLOBBER
75976 /* '>|' */

75977 /* The following are the reserved words. */

75978 %token If Then Else Elif Fi Do Done
75979 /* 'if' 'then' 'else' 'elif' 'fi' 'do' 'done' */

75980 %token Case Esac While Until For
75981 /* 'case' 'esac' 'while' 'until' 'for' */

75982 /* These are reserved words, not operator tokens, and are
75983 recognized when reserved words are recognized. */

75984 %token Lbrace Rbrace Bang
75985 /* '{' '}' '!' */

75986 %token In
75987 /* 'in' */

```



```

75988      /* -----
75989         The Grammar
75990         ----- */
75991      %start program
75992      %%
75993      program      : linebreak complete_commands linebreak
75994                   | linebreak
75995                   ;
75996      complete_commands: complete_commands newline_list complete_command
75997                   |                               complete_command
75998                   ;
75999      complete_command : list separator_op
76000                   | list
76001                   ;
76002      list           : list separator_op and_or
76003                   |                               and_or
76004                   ;
76005      and_or        :                               pipeline
76006                   | and_or AND_IF linebreak pipeline
76007                   | and_or OR_IF  linebreak pipeline
76008                   ;
76009      pipeline      :       pipe_sequence
76010                   | Bang pipe_sequence
76011                   ;
76012      pipe_sequence :                               command
76013                   | pipe_sequence '|' linebreak command
76014                   ;
76015      command       : simple_command
76016                   | compound_command
76017                   | compound_command redirect_list
76018                   | function_definition
76019                   ;
76020      compound_command : brace_group
76021                   | subshell
76022                   | for_clause
76023                   | case_clause
76024                   | if_clause
76025                   | while_clause
76026                   | until_clause
76027                   ;
76028      subshell      : '(' compound_list ')'
76029                   ;
76030      compound_list : linebreak term
76031                   | linebreak term separator
76032                   ;
76033      term          : term separator and_or
76034                   |                               and_or
76035                   ;
76036      for_clause    : For name                               do_group
76037                   | For name                               sequential_sep do_group
76038                   | For name linebreak in                 sequential_sep do_group
76039                   | For name linebreak in wordlist sequential_sep do_group
76040                   ;

```

```

76041     name           : NAME                               /* Apply rule 5 */
76042     ;
76043     in               : In                                 /* Apply rule 6 */
76044     ;
76045     wordlist         : wordlist WORD
76046     |                 | WORD
76047     ;
76048     case_clause      : Case WORD linebreak in linebreak case_list Esac
76049     |                 | Case WORD linebreak in linebreak case_list_ns Esac
76050     |                 | Case WORD linebreak in linebreak Esac
76051     ;
76052     case_list_ns     : case_list case_item_ns
76053     |                 | case_item_ns
76054     ;
76055     case_list        : case_list case_item
76056     |                 | case_item
76057     ;
76058     case_item_ns     : pattern ')' linebreak
76059     |                 | pattern ')' compound_list
76060     |                 | '(' pattern ')' linebreak
76061     |                 | '(' pattern ')' compound_list
76062     ;
76063     case_item        : pattern ')' linebreak DSEMI linebreak
76064     |                 | pattern ')' compound_list DSEMI linebreak
76065     |                 | '(' pattern ')' linebreak DSEMI linebreak
76066     |                 | '(' pattern ')' compound_list DSEMI linebreak
76067     ;
76068     pattern          : WORD                               /* Apply rule 4 */
76069     |                 | pattern '|' WORD                 /* Do not apply rule 4 */
76070     ;
76071     if_clause        : If compound_list Then compound_list else_part Fi
76072     |                 | If compound_list Then compound_list Fi
76073     ;
76074     else_part        : Elif compound_list Then compound_list
76075     |                 | Elif compound_list Then compound_list else_part
76076     |                 | Else compound_list
76077     ;
76078     while_clause     : While compound_list do_group
76079     ;
76080     until_clause     : Until compound_list do_group
76081     ;
76082     function_definition : fname '(' ')' linebreak function_body
76083     ;
76084     function_body    : compound_command                 /* Apply rule 9 */
76085     |                 | compound_command redirect_list /* Apply rule 9 */
76086     ;
76087     fname           : NAME                               /* Apply rule 8 */
76088     ;
76089     brace_group      : Lbrace compound_list Rbrace
76090     ;
76091     do_group         : Do compound_list Done             /* Apply rule 6 */
76092     ;
76093     simple_command   : cmd_prefix cmd_word cmd_suffix

```

```

76094 | cmd_prefix cmd_word
76095 | cmd_prefix
76096 | cmd_name cmd_suffix
76097 | cmd_name
76098 ;
76099 cmd_name : WORD /* Apply rule 7a */
76100 ;
76101 cmd_word : WORD /* Apply rule 7b */
76102 ;
76103 cmd_prefix : io_redirect
76104 | cmd_prefix io_redirect
76105 | ASSIGNMENT_WORD
76106 | cmd_prefix ASSIGNMENT_WORD
76107 ;
76108 cmd_suffix : io_redirect
76109 | cmd_suffix io_redirect
76110 | WORD
76111 | cmd_suffix WORD
76112 ;
76113 redirect_list : io_redirect
76114 | redirect_list io_redirect
76115 ;
76116 io_redirect : io_file
76117 | IO_NUMBER io_file
76118 | io_here
76119 | IO_NUMBER io_here
76120 ;
76121 io_file : '<' filename
76122 | LESSAND filename
76123 | '>' filename
76124 | GREATAND filename
76125 | DGREAT filename
76126 | LESSGREAT filename
76127 | CLOBBER filename
76128 ;
76129 filename : WORD /* Apply rule 2 */
76130 ;
76131 io_here : DLESS here_end
76132 | DLESSDASH here_end
76133 ;
76134 here_end : WORD /* Apply rule 3 */
76135 ;
76136 newline_list : NEWLINE
76137 | newline_list NEWLINE
76138 ;
76139 linebreak : newline_list
76140 | /* empty */
76141 ;
76142 separator_op : '&'
76143 | ';'
76144 ;
76145 separator : separator_op linebreak
76146 | newline_list

```

```

76147          ;
76148 sequential_sep : ';' linebreak
76149                | newline_list
76150          ;

```

76151 2.11 Signals and Error Handling

76152 If job control is disabled (see the description of *set -m*) when the shell executes an asynchronous
 76153 list, the commands in the list shall inherit from the shell a signal action of ignored (SIG_IGN) for
 76154 the SIGINT and SIGQUIT signals. In all other cases, commands executed by the shell shall
 76155 inherit the same signal actions as those inherited by the shell from its parent unless a signal
 76156 action is modified by the *trap* special built-in (see *trap*)

76157 When a signal for which a trap has been set is received while the shell is waiting for the
 76158 completion of a utility executing a foreground command, the trap associated with that signal
 76159 shall not be executed until after the foreground command has completed. When the shell is
 76160 waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a
 76161 signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit
 76162 status >128, immediately after which the trap associated with that signal shall be taken.

76163 If multiple signals are pending for the shell for which there are associated trap actions, the order
 76164 of execution of trap actions is unspecified.

76165 2.12 Shell Execution Environment

76166 A shell execution environment consists of the following:

76167 Open files inherited upon invocation of the shell, plus open files controlled by *exec*

76168 Working directory as set by *cd*

76169 File creation mask set by *umask*

76170 XSI File size limit as set by *ulimit*

76171 Current traps set by *trap*

76172 Shell parameters that are set by variable assignment (see the *set* special built-in) or from
 76173 the System Interfaces volume of POSIX.1-2017 environment inherited by the shell when it
 76174 begins (see the *export* special built-in)

76175 Shell functions; see [Section 2.9.5](#)

76176 Options turned on at invocation or by *set*

76177 Process IDs of the last commands in asynchronous lists known to this shell environment;
 76178 see [Section 2.9.3.1](#)

76179 Shell aliases; see [Section 2.3.1](#)

76180 Utilities other than the special built-ins (see [Section 2.14](#)) shall be invoked in a separate
 76181 environment that consists of the following. The initial value of these objects shall be the same as
 76182 that for the parent shell, except as noted below.

76183 Open files inherited on invocation of the shell, open files controlled by the *exec* special
 76184 built-in plus any modifications, and additions specified by any redirections to the utility

76185 Current working directory

76186 File creation mask

76187 If the utility is a shell script, traps caught by the shell shall be set to the default values and
 76188 traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell
 76189 script, the trap actions (default or ignore) shall be mapped into the appropriate signal
 76190 handling actions for the utility

76191 Variables with the *export* attribute, along with those explicitly exported for the duration of
 76192 the command, shall be passed to the utility environment variables

76193 The environment of the shell process shall not be changed by the utility unless explicitly
 76194 specified by the utility description (for example, *cd* and *umask*).

76195 A subshell environment shall be created as a duplicate of the shell environment, except that
 76196 signal traps that are not being ignored shall be set to the default action. Changes made to the
 76197 subshell environment shall not affect the shell environment. Command substitution, commands
 76198 that are grouped with parentheses, and asynchronous lists shall be executed in a subshell
 76199 environment. Additionally, each command of a multi-command pipeline is in a subshell
 76200 environment; as an extension, however, any or all commands in a pipeline may be executed in
 76201 the current environment. All other commands shall be executed in the current shell
 76202 environment.

76203 2.13 Pattern Matching Notation

76204 The pattern matching notation described in this section is used to specify patterns for matching
 76205 strings in the shell. Historically, pattern matching notation is related to, but slightly different
 76206 from, the regular expression notation described in XBD [Chapter 9](#). For this reason, the
 76207 description of the rules for this pattern matching notation are based on the description of regular
 76208 expression notation, modified to account for the differences.

76209 2.13.1 Patterns Matching a Single Character

76210 The following patterns matching a single character shall match a single character: ordinary
 76211 characters, special pattern characters, and pattern bracket expressions. The pattern bracket
 76212 expression also shall match a single collating element. A `<backslash>` character shall escape the
 76213 following character. The escaping `<backslash>` shall be discarded. If a pattern ends with an
 76214 unescaped `<backslash>`, it is unspecified whether the pattern does not match anything or the
 76215 pattern is treated as invalid.

76216 An ordinary character is a pattern that shall match itself. It can be any character in the supported
 76217 character set except for NUL, those special shell characters in [Section 2.2](#) that require quoting,
 76218 and the following three special pattern characters. Matching shall be based on the bit pattern
 76219 used for encoding the character, not on the graphic representation of the character. If any
 76220 character (ordinary, shell special, or pattern special) is quoted, that pattern shall match the
 76221 character itself. The shell special characters always require quoting.

76222 When unquoted and outside a bracket expression, the following three characters shall have
 76223 special meaning in the specification of patterns:

- 76224 ? A <question-mark> is a pattern that shall match any character.
- 76225 * An <asterisk> is a pattern that shall match multiple characters, as described in [Section](#)
76226 [2.13.2](#).
- 76227 [If an open bracket introduces a bracket expression as in XBD [Section 9.3.5](#), except that the
76228 <exclamation-mark> character ('!') shall replace the <circumflex> character ('^') in its
76229 role in a non-matching list in the regular expression notation, it shall introduce a pattern
76230 bracket expression. A bracket expression starting with an unquoted <circumflex> character
76231 produces unspecified results. Otherwise, '[' shall match the character itself.
- 76232 When pattern matching is used where shell quote removal is not performed (such as in the
76233 argument to the *find -name* primary when *find* is being called using one of the *exec* functions as
76234 defined in the System Interfaces volume of POSIX.1-2017, or in the *pattern* argument to the
76235 *fnmatch()* function), special characters can be escaped to remove their special meaning by
76236 preceding them with a <backslash> character. This escaping <backslash> is discarded. The
76237 sequence "\\\" represents one literal <backslash>. All of the requirements and effects of quoting
76238 on ordinary, shell special, and special pattern characters shall apply to escaping in this context.

76239 2.13.2 Patterns Matching Multiple Characters

76240 The following rules are used to construct patterns matching multiple characters from patterns
76241 matching a single character:

- 76242 1. The <asterisk> ('*') is a pattern that shall match any string, including the null string.
- 76243 2. The concatenation of patterns matching a single character is a valid pattern that shall
76244 match the concatenation of the single characters or collating elements matched by each of
76245 the concatenated patterns.
- 76246 3. The concatenation of one or more patterns matching a single character with one or more
76247 <asterisk> characters is a valid pattern. In such patterns, each <asterisk> shall match a
76248 string of zero or more characters, matching the greatest possible number of characters
76249 that still allows the remainder of the pattern to match the string.

76250 2.13.3 Patterns Used for Filename Expansion

76251 The rules described so far in [Section 2.13.1](#) and [Section 2.13.2](#) are qualified by the following rules
76252 that apply when pattern matching notation is used for filename expansion:

- 76253 1. The <slash> character in a pathname shall be explicitly matched by using one or more
76254 <slash> characters in the pattern; it shall neither be matched by the <asterisk> or
76255 <question-mark> special characters nor by a bracket expression. <slash> characters in the
76256 pattern shall be identified before bracket expressions; thus, a <slash> cannot be included
76257 in a pattern bracket expression used for filename expansion. If a <slash> character is
76258 found following an unescaped <left-square-bracket> character before a corresponding
76259 <right-square-bracket> is found, the open bracket shall be treated as an ordinary
76260 character. For example, the pattern "a[b/c]d" does not match such pathnames as **abd**
76261 or **a/d**. It only matches a pathname of literally **a[b/c]d**.
- 76262 2. If a filename begins with a <period> ('.'), the <period> shall be explicitly matched by
76263 using a <period> as the first character of the pattern or immediately following a <slash>
76264 character. The leading <period> shall not be matched by:

- 76265 The <asterisk> or <question-mark> special characters
- 76266 A bracket expression containing a non-matching list, such as "[!a]", a range
76267 expression, such as "[%-0]", or a character class expression, such as
76268 "[[:punct:]]"
- 76269 It is unspecified whether an explicit <period> in a bracket expression matching list, such
76270 as "[.abc]", can match a leading <period> in a filename.
- 76271 3. Specified patterns shall be matched against existing filenames and pathnames, as
76272 appropriate. Each component that contains a pattern character shall require read
76273 permission in the directory containing that component. Any component, except the last,
76274 that does not contain a pattern character shall require search permission. For example,
76275 given the pattern:
- 76276 /foo/bar/x*/bam
- 76277 search permission is needed for directories / and **foo**, search and read permissions are
76278 needed for directory **bar**, and search permission is needed for each **x*** directory. If the
76279 pattern matches any existing filenames or pathnames, the pattern shall be replaced with
76280 those filenames and pathnames, sorted according to the collating sequence in effect in the
76281 current locale. If this collating sequence does not have a total ordering of all characters
76282 (see XBD [Section 7.3.2](#), on page 147), any filenames or pathnames that collate equally
76283 should be further compared byte-by-byte using the collating sequence for the POSIX
76284 locale.
- 76285 **Note:** A future version of this standard may require the byte-by-byte further comparison
76286 described above.
- 76287 If the pattern contains an open bracket ('[') that does not introduce a bracket expression
76288 as in XBD [Section 9.3.5](#), it is unspecified whether other unquoted pattern matching
76289 characters within the same slash-delimited component of the pattern retain their special
76290 meanings or are treated as ordinary characters. For example, the pattern "a*[/b*" may
76291 match all filenames beginning with 'b' in the directory "a*[" or it may match all
76292 filenames beginning with 'b' in all directories with names beginning with 'a' and
76293 ending with '['.
- 76294 If the pattern does not match any existing filenames or pathnames, the pattern string shall
76295 be left unchanged.

76296 2.14 Special Built-In Utilities

- 76297 The following ``special built-in'' utilities shall be supported in the shell command language. The
76298 output of each command, if any, shall be written to standard output, subject to the normal
76299 redirection and piping possible with all commands.
- 76300 The term ``built-in'' implies that the shell can execute the utility directly and does not need to
76301 search for it. An implementation may choose to make any utility a built-in; however, the special
76302 built-in utilities described here differ from regular built-in utilities in two respects:
- 76303 1. An error in a special built-in utility may cause a shell executing that utility to abort, while
76304 an error in a regular built-in utility shall not cause a shell executing that utility to abort.
76305 (See [Section 2.8.1](#) for the consequences of errors on interactive and non-interactive shells.)
76306 If a special built-in utility encountering an error does not abort the shell, its exit value
76307 shall be non-zero.

76308 2. As described in [Section 2.9.1](#), variable assignments preceding the invocation of a special
76309 built-in utility remain in effect after the built-in completes; this shall not be the case with a
76310 regular built-in or other utility.

76311 The special built-in utilities in this section need not be provided in a manner accessible via the
76312 *exec* family of functions defined in the System Interfaces volume of POSIX.1-2017.

76313 Some of the special built-ins are described as conforming to XBD [Section 12.2](#). For those that are
76314 not, the requirement in [Section 1.4](#) that "--" be recognized as a first argument to be discarded
76315 does not apply and a conforming application shall not use that argument.

76316 **NAME**

76317 break — exit from for, while, or until loop

76318 **SYNOPSIS**76319 break [*n*]76320 **DESCRIPTION**

76321 If *n* is specified, the *break* utility shall exit from the *n*th enclosing **for**, **while**, or **until** loop. If *n* is
76322 not specified, *break* shall behave as if *n* was specified as 1. Execution shall continue with the
76323 command immediately following the exited loop. The value of *n* is a positive decimal integer. If
76324 *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be exited. If
76325 there is no enclosing loop, the behavior is unspecified.

76326 A loop shall enclose a *break* or *continue* command if the loop lexically encloses the command. A
76327 loop lexically encloses a *break* or *continue* command if the command is:

76328 Executing in the same execution environment (see [Section 2.12](#)) as the compound-list of the
76329 loop's do-group (see [Section 2.10.2](#)), and

76330 Contained in a compound-list associated with the loop (either in the compound-list of the
76331 loop's do-group or, if the loop is a **while** or **until** loop, in the compound-list following the
76332 **while** or **until** reserved word), and

76333 Not in the body of a function whose function definition command (see [Section 2.9.5](#)) is
76334 contained in a compound-list associated with the loop.

76335 If *n* is greater than the number of lexically enclosing loops and there is a non-lexically enclosing
76336 loop in progress in the same execution environment as the *break* or *continue* command, it is
76337 unspecified whether that loop encloses the command.

76338 **OPTIONS**

76339 None.

76340 **OPERANDS**

76341 See the DESCRIPTION.

76342 **STDIN**

76343 Not used.

76344 **INPUT FILES**

76345 None.

76346 **ENVIRONMENT VARIABLES**

76347 None.

76348 **ASYNCHRONOUS EVENTS**

76349 Default.

76350 **STDOUT**

76351 Not used.

76352 **STDERR**

76353 The standard error shall be used only for diagnostic messages.

76354 **OUTPUT FILES**

76355 None.

76356 **EXTENDED DESCRIPTION**

76357 None.

76358 **EXIT STATUS**

76359 0 Successful completion.

76360 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.76361 **CONSEQUENCES OF ERRORS**

76362 Default.

76363 **APPLICATION USAGE**

76364 None.

76365 **EXAMPLES**

```

76366     for i in *
76367     do
76368         if test -d "$i"
76369         then break
76370         fi
76371     done

```

76372 The results of running the following example are unspecified: there are two loops in progress
 76373 when the *break* command is executed, and they are in the same execution environment, but
 76374 neither loop is lexically enclosing the *break* command. (There are no loops lexically enclosing the
 76375 *continue* commands, either.)

```

76376     foo() {
76377         for j in 1 2; do
76378             echo 'break 2' >/tmp/do_break
76379             echo " sourcing /tmp/do_break ($j)..."
76380             # the behavior of the break from running the following command
76381             # results in unspecified behavior:
76382             . /tmp/do_break

76383             do_continue() { continue 2; }
76384             echo " running do_continue ($j)..."
76385             # the behavior of the continue in the following function call
76386             # results in unspecified behavior (if execution reaches this
76387             # point):
76388             do_continue

76389             trap 'continue 2' USR1
76390             echo " sending SIGUSR1 to self ($j)..."
76391             # the behavior of the continue in the trap invoked from the
76392             # following signal results in unspecified behavior (if
76393             # execution reaches this point):
76394             kill -s USR1 $$
76395             sleep 1
76396         done
76397     }
76398     for i in 1 2; do
76399         echo "running foo ($i)..."
76400         foo
76401     done

```

76402 **RATIONALE**

76403 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer
76404 to a label associated with the appropriate loop as a preferable alternative to the *n* method.
76405 However, this volume of POSIX.1-2017 does reserve the name space of command names ending
76406 with a <colon>. It is anticipated that a future implementation could take advantage of this and
76407 provide something like:

```
76408 outofloop: for i in a b c d e  
76409 do  
76410     for j in 0 1 2 3 4 5 6 7 8 9  
76411     do  
76412         if test -r "${i}${j}"  
76413         then break outofloop  
76414         fi  
76415     done  
76416 done
```

76417 and that this might be standardized after implementation experience is achieved.

76418 **FUTURE DIRECTIONS**

76419 None.

76420 **SEE ALSO**

76421 [Section 2.14](#)

76422 **CHANGE HISTORY**76423 **Issue 6**

76424 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76425 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76426 behavior is intended.

76427 **Issue 7**

76428 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0046 [842] is applied.

76429 **NAME**

76430 colon ‡'null utility

76431 **SYNOPSIS**76432 : [*argument...*]76433 **DESCRIPTION**76434 This utility shall only expand command *arguments*. It is used when a command is needed, as in
76435 the **then** condition of an **if** command, but nothing is to be done by the command.76436 **OPTIONS**

76437 None.

76438 **OPERANDS**

76439 See the DESCRIPTION.

76440 **STDIN**

76441 Not used.

76442 **INPUT FILES**

76443 None.

76444 **ENVIRONMENT VARIABLES**

76445 None.

76446 **ASYNCHRONOUS EVENTS**

76447 Default.

76448 **STDOUT**

76449 Not used.

76450 **STDERR**

76451 The standard error shall be used only for diagnostic messages.

76452 **OUTPUT FILES**

76453 None.

76454 **EXTENDED DESCRIPTION**

76455 None.

76456 **EXIT STATUS**

76457 Zero.

76458 **CONSEQUENCES OF ERRORS**

76459 Default.

76460 **APPLICATION USAGE**

76461 None.

76462 **EXAMPLES**76463 : \${X=abc}
76464 if false
76465 then :
76466 else echo \$X
76467 fi
76468 **abc**76469 As with any of the special built-ins, the null utility can also have variable assignments and
76470 redirections associated with it, such as:

76471 x=y : > z

76472 which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates
76473 or truncates file *z*.

76474 **RATIONALE**

76475 None.

76476 **FUTURE DIRECTIONS**

76477 None.

76478 **SEE ALSO**

76479 [Section 2.14](#)

76480 **CHANGE HISTORY**

76481 **Issue 6**

76482 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76483 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76484 behavior is intended.

76485 **Issue 7**

76486 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76487 **NAME**

76488 continue ‡continue forwhile, or until loop

76489 **SYNOPSIS**76490 continue [*n*]76491 **DESCRIPTION**

76492 If *n* is specified, the *continue* utility shall return to the top of the *n*th enclosing **for**, **while**, or **until**
76493 loop. If *n* is not specified, *continue* shall behave as if *n* was specified as 1. Returning to the top of
76494 the loop involves repeating the condition list of a **while** or **until** loop or performing the next
76495 assignment of a **for** loop, and re-executing the loop if appropriate.

76496 The value of *n* is a positive decimal integer. If *n* is greater than the number of enclosing loops,
76497 the outermost enclosing loop shall be used. If there is no enclosing loop, the behavior is
76498 unspecified.

76499 The meaning of “enclosing” shall be as specified in the description of the *break* utility.

76500 **OPTIONS**

76501 None.

76502 **OPERANDS**

76503 See the DESCRIPTION.

76504 **STDIN**

76505 Not used.

76506 **INPUT FILES**

76507 None.

76508 **ENVIRONMENT VARIABLES**

76509 None.

76510 **ASYNCHRONOUS EVENTS**

76511 Default.

76512 **STDOUT**

76513 Not used.

76514 **STDERR**

76515 The standard error shall be used only for diagnostic messages.

76516 **OUTPUT FILES**

76517 None.

76518 **EXTENDED DESCRIPTION**

76519 None.

76520 **EXIT STATUS**

76521 0 Successful completion.

76522 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.76523 **CONSEQUENCES OF ERRORS**

76524 Default.

76525 **APPLICATION USAGE**

76526 None.

76527 **EXAMPLES**

```
76528     for i in *
76529     do
76530         if test -d "$i"
76531         then continue
76532         fi
76533         printf "%s" is not a directory.\n' "$i"
76534     done
```

76535 **RATIONALE**

76536 None.

76537 **FUTURE DIRECTIONS**

76538 None.

76539 **SEE ALSO**76540 [Section 2.14](#)76541 **CHANGE HISTORY**76542 **Issue 6**

76543 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76544 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76545 behavior is intended.

76546 **Issue 7**76547 The example is changed to use the *printf* utility rather than *echo*.

76548 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0046 [842] is applied.

76549 **NAME**

76550 dot — execute commands in the current environment

76551 **SYNOPSIS**

76552 . *file*

76553 **DESCRIPTION**

76554 The shell shall execute commands from the *file* in the current environment.

76555 If *file* does not contain a <slash>, the shell shall use the search path specified by *PATH* to find the
76556 directory containing *file*. Unlike normal command search, however, the file searched for by the
76557 *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort;
76558 an interactive shell shall write a diagnostic message to standard error, but this condition shall
76559 not be considered a syntax error.

76560 **OPTIONS**

76561 None.

76562 **OPERANDS**

76563 See the DESCRIPTION.

76564 **STDIN**

76565 Not used.

76566 **INPUT FILES**

76567 See the DESCRIPTION.

76568 **ENVIRONMENT VARIABLES**

76569 See the DESCRIPTION.

76570 **ASYNCHRONOUS EVENTS**

76571 Default.

76572 **STDOUT**

76573 Not used.

76574 **STDERR**

76575 The standard error shall be used only for diagnostic messages.

76576 **OUTPUT FILES**

76577 None.

76578 **EXTENDED DESCRIPTION**

76579 None.

76580 **EXIT STATUS**

76581 If no readable file was found or if the commands in the file could not be parsed, and the shell is
76582 interactive (and therefore does not abort; see [Section 2.8.1](#)), the exit status shall be non-zero.
76583 Otherwise, return the value of the last command executed, or a zero exit status if no command is
76584 executed.

76585 **CONSEQUENCES OF ERRORS**

76586 Default.

76587 **APPLICATION USAGE**

76588 None.

76589 **EXAMPLES**

```
76590     cat foobar
76591     foo=hello bar=world
76592     . ./foobar
76593     echo $foo $bar
76594     hello world
```

76595 **RATIONALE**

76596 Some older implementations searched the current directory for the *file*, even if the value of *PATH*
76597 disallowed it. This behavior was omitted from this volume of POSIX.1-2017 due to concerns
76598 about introducing the susceptibility to trojan horses that the user might be trying to avoid by
76599 leaving **dot** out of *PATH*.

76600 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.
76601 This is a valid extension that allows a *dot* script to behave identically to a function.

76602 **FUTURE DIRECTIONS**

76603 None.

76604 **SEE ALSO**76605 [Section 2.14, *return*](#)76606 **CHANGE HISTORY**76607 **Issue 6**

76608 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76609 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76610 behavior is intended.

76611 **Issue 7**

76612 SD5-XCU-ERN-164 is applied.

76613 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0038 [114] and XCU/TC1-2008/0039
76614 [214] are applied.

76615 **NAME**

76616 eval — construct command by concatenating arguments

76617 **SYNOPSIS**

76618 eval [*argument...*]

76619 **DESCRIPTION**

76620 The *eval* utility shall construct a command by concatenating *arguments* together, separating each
76621 with a <space> character. The constructed command shall be read and executed by the shell.

76622 **OPTIONS**

76623 None.

76624 **OPERANDS**

76625 See the DESCRIPTION.

76626 **STDIN**

76627 Not used.

76628 **INPUT FILES**

76629 None.

76630 **ENVIRONMENT VARIABLES**

76631 None.

76632 **ASYNCHRONOUS EVENTS**

76633 Default.

76634 **STDOUT**

76635 Not used.

76636 **STDERR**

76637 The standard error shall be used only for diagnostic messages.

76638 **OUTPUT FILES**

76639 None.

76640 **EXTENDED DESCRIPTION**

76641 None.

76642 **EXIT STATUS**

76643 If there are no *arguments*, or only null *arguments*, *eval* shall return a zero exit status; otherwise, it
76644 shall return the exit status of the command defined by the string of concatenated *arguments*
76645 separated by <space> characters, or a non-zero exit status if the concatenation could not be
76646 parsed as a command and the shell is interactive (and therefore did not abort).

76647 **CONSEQUENCES OF ERRORS**

76648 Default.

76649 **APPLICATION USAGE**

76650 Since *eval* is not required to recognize the "--" end of options delimiter, in cases where the
76651 argument(s) to *eval* might begin with '-' it is recommended that the first argument is prefixed
76652 by a string that will not alter the commands to be executed, such as a <space> character:

76653 eval " \$commands "

76654 or:

76655 eval " \$(some_command) "

76656 EXAMPLES

```
76657     foo=10 x=foo
76658     y='$'$x
76659     echo $y
76660     $foo
76661     eval y='$'$x
76662     echo $y
76663     10
```

76664 RATIONALE

76665 This standard allows, but does not require, *eval* to recognize "--". Although this means
76666 applications cannot use "--" to protect against options supported as an extension (or errors
76667 reported for unsupported options), the nature of the *eval* utility is such that other means can be
76668 used to provide this protection (see APPLICATION USAGE above).

76669 FUTURE DIRECTIONS

76670 None.

76671 SEE ALSO

76672 [Section 2.14](#)

76673 CHANGE HISTORY**76674 Issue 6**

76675 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76676 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76677 behavior is intended.

76678 Issue 7

76679 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76680 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0040 [114], XCU/TC1-2008/0041 [163],
76681 and XCU/TC1-2008/0042 [163] are applied.

76682 **NAME**

76683 exec ‡execute commands and open, close, or copy file descriptors

76684 **SYNOPSIS**76685 exec [*command* [*argument...*]]76686 **DESCRIPTION**76687 The *exec* utility shall open, close, and/or copy file descriptors as specified by any redirections as
76688 part of the command.76689 If *exec* is specified without *command* or *arguments*, and any file descriptors with numbers greater
76690 than 2 are opened with associated redirection statements, it is unspecified whether those file
76691 descriptors remain open when the shell invokes another utility. Scripts concerned that child
76692 shells could misuse open file descriptors can always close them explicitly, as shown in one of the
76693 following examples.76694 If *exec* is specified with *command*, it shall replace the shell with *command* without creating a new
76695 process. If *arguments* are specified, they shall be arguments to *command*. Redirection affects the
76696 current shell execution environment.76697 **OPTIONS**

76698 None.

76699 **OPERANDS**

76700 See the DESCRIPTION.

76701 **STDIN**

76702 Not used.

76703 **INPUT FILES**

76704 None.

76705 **ENVIRONMENT VARIABLES**

76706 None.

76707 **ASYNCHRONOUS EVENTS**

76708 Default.

76709 **STDOUT**

76710 Not used.

76711 **STDERR**

76712 The standard error shall be used only for diagnostic messages.

76713 **OUTPUT FILES**

76714 None.

76715 **EXTENDED DESCRIPTION**

76716 None.

76717 **EXIT STATUS**76718 If *command* is specified, *exec* shall not return to the shell; rather, the exit status of the process shall
76719 be the exit status of the program implementing *command*, which overlaid the shell. If *command* is
76720 not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the
76721 exit status shall be 126. If a redirection error occurs (see [Section 2.8.1](#)), the shell shall exit with a
76722 value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

76723 **CONSEQUENCES OF ERRORS**

76724 Default.

76725 **APPLICATION USAGE**

76726 None.

76727 **EXAMPLES**76728 Open *readfile* as file descriptor 3 for reading:76729 `exec 3< readfile`76730 Open *writefile* as file descriptor 4 for writing:76731 `exec 4> writefile`

76732 Make file descriptor 5 a copy of file descriptor 0:

76733 `exec 5<&0`

76734 Close file descriptor 3:

76735 `exec 3<&-`76736 Cat the file **maggie** by replacing the current shell with the *cat* utility:76737 `exec cat maggie`76738 **RATIONALE**

76739 Most historical implementations were not conformant in that:

76740 `foo=bar exec cmd`76741 did not pass **foo** to **cmd**.76742 **FUTURE DIRECTIONS**

76743 None.

76744 **SEE ALSO**76745 [Section 2.14](#)76746 **CHANGE HISTORY**76747 **Issue 6**76748 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76749 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76750 behavior is intended.76751 **Issue 7**

76752 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

76753 **NAME**

76754 exit ‡'cause the shell to exit

76755 **SYNOPSIS**

76756 exit [n]

76757 **DESCRIPTION**

76758 The *exit* utility shall cause the shell to exit from its current execution environment with the exit
76759 status specified by the unsigned decimal integer *n*. If the current execution environment is a
76760 subshell environment, the shell shall exit from the subshell environment with the specified exit
76761 status and continue in the environment from which that subshell environment was invoked;
76762 otherwise, the shell utility shall terminate with the specified exit status. If *n* is specified, but its
76763 value is not between 0 and 255 inclusively, the exit status is undefined.

76764 A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is
76765 invoked in that *trap* itself, in which case the shell shall exit immediately.

76766 **OPTIONS**

76767 None.

76768 **OPERANDS**

76769 See the DESCRIPTION.

76770 **STDIN**

76771 Not used.

76772 **INPUT FILES**

76773 None.

76774 **ENVIRONMENT VARIABLES**

76775 None.

76776 **ASYNCHRONOUS EVENTS**

76777 Default.

76778 **STDOUT**

76779 Not used.

76780 **STDERR**

76781 The standard error shall be used only for diagnostic messages.

76782 **OUTPUT FILES**

76783 None.

76784 **EXTENDED DESCRIPTION**

76785 None.

76786 **EXIT STATUS**

76787 The exit status shall be *n*, if specified, except that the behavior is unspecified if *n* is not an
76788 unsigned decimal integer or is greater than 255. Otherwise, the value shall be the exit value of
76789 the last command executed, or zero if no command was executed. When *exit* is executed in a *trap*
76790 action, the last command is considered to be the command that executed immediately preceding
76791 the *trap* action.

76792 **CONSEQUENCES OF ERRORS**

76793 Default.

76794 APPLICATION USAGE

76795 None.

76796 EXAMPLES

76797 Exit with a *true* value:

```
76798 exit 0
```

76799 Exit with a *false* value:

```
76800 exit 1
```

76801 Propagate error handling from within a subshell:

```
76802 (  
76803     command1 || exit 1  
76804     command2 || exit 1  
76805     exec command3  
76806 ) > outputfile || exit 1  
76807 echo "outputfile created successfully"
```

76808 RATIONALE

76809 As explained in other sections, certain exit status values have been reserved for special uses and
76810 should be used by applications only for those purposes:

76811 126 A file to be executed was found, but it was not an executable utility.

76812 127 A utility to be executed was not found.

76813 >128 A command was interrupted by a signal.

76814 The behavior of *exit* when given an invalid argument or unknown option is unspecified, because
76815 of differing practices in the various historical implementations. A value larger than 255 might be
76816 truncated by the shell, and be unavailable even to a parent process that uses *waitid()* to get the
76817 full exit value. It is recommended that implementations that detect any usage error should cause
76818 a non-zero exit status (or, if the shell is interactive and the error does not cause the shell to abort,
76819 store a non-zero value in "\$?"), but even this was not done historically in all shells.

76820 FUTURE DIRECTIONS

76821 None.

76822 SEE ALSO

76823 [Section 2.14](#)

76824 CHANGE HISTORY**76825 Issue 6**

76826 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76827 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
76828 behavior is intended.

76829 Issue 7

76830 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0047 [717], XCU/TC2-2008/0048
76831 [960], XCU/TC2-2008/0049 [717], and XCU/TC2-2008/0050 [960] are applied.

76832 **NAME**76833 `export` `†`'set the export attribute for variables76834 **SYNOPSIS**76835 `export name [=word] . . .`76836 `export -p`76837 **DESCRIPTION**

76838 The shell shall give the *export* attribute to the variables corresponding to the specified *names*,
76839 which shall cause them to be in the environment of subsequently executed commands. If the
76840 name of a variable is followed by *=word*, then the value of that variable shall be set to *word*.

76841 The *export* special built-in shall support XBD [Section 12.2](#).

76842 When `-p` is specified, *export* shall write to the standard output the names and values of all
76843 exported variables, in the following format:

76844 `"export %s=%s\n", <name>, <value>`76845 if *name* is set, and:76846 `"export %s\n", <name>`76847 if *name* is unset.

76848 The shell shall format the output, including the proper use of quoting, so that it is suitable for
76849 reinput to the shell as commands that achieve the same exporting results, except:

- 76850 1. Read-only variables with values cannot be reset.
- 76851 2. Variables that were unset at the time they were output need not be reset to the unset state
76852 if a value is assigned to the variable between the time the state was saved and the time at
76853 which the saved output is reinput to the shell.

76854 When no arguments are given, the results are unspecified.

76855 **OPTIONS**

76856 See the DESCRIPTION.

76857 **OPERANDS**

76858 See the DESCRIPTION.

76859 **STDIN**

76860 Not used.

76861 **INPUT FILES**

76862 None.

76863 **ENVIRONMENT VARIABLES**

76864 None.

76865 **ASYNCHRONOUS EVENTS**

76866 Default.

76867 **STDOUT**

76868 See the DESCRIPTION.

76869 **STDERR**

76870 The standard error shall be used only for diagnostic messages.

76871 **OUTPUT FILES**

76872 None.

76873 **EXTENDED DESCRIPTION**

76874 None.

76875 **EXIT STATUS**76876 0 All *name* operands were successfully exported.76877 >0 At least one *name* could not be exported, or the **-p** option was specified and an error
76878 occurred.76879 **CONSEQUENCES OF ERRORS**

76880 Default.

76881 **APPLICATION USAGE**76882 Note that, unless *X* was previously marked readonly, the value of "\$?" after:76883 `export X=$(false)`76884 will be 0 (because *export* successfully set *X* to the empty string) and that execution continues,
76885 even if *set -e* is in effect. In order to detect command substitution failures, a user must separate
76886 the assignment from the export, as in:76887 `X=$(false)`76888 `export X`76889 **EXAMPLES**76890 Export *PWD* and *HOME* variables:76891 `export PWD HOME`76892 Set and export the *PATH* variable:76893 `export PATH=/local/bin:$PATH`

76894 Save and restore all exported variables:

76895 `export -p > temp-file`76896 `unset a lot of variables`76897 `... processing`76898 `. temp-file`76899 **RATIONALE**76900 Some historical shells use the no-argument case as the functional equivalent of what is required
76901 here with **-p**. This feature was left unspecified because it is not historical practice in all shells,
76902 and some scripts may rely on the now-unspecified results on their implementations. Attempts to
76903 specify the **-p** output as the default case were unsuccessful in achieving consensus. The **-p**
76904 option was added to allow portable access to the values that can be saved and then later restored
76905 using; for example, a *dot* script.76906 **FUTURE DIRECTIONS**

76907 None.

76908 **SEE ALSO**76909 [Section 2.14](#)76910 [XBD Section 12.2](#)

76911 **CHANGE HISTORY**76912 **Issue 6**

76913 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

76914 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
76915 sections use terms as described in the Utility Description Defaults (Section 1.4). No change in
76916 behavior is intended.

76917 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to
76918 the end of the first paragraph of the DESCRIPTION: ``If the name of a variable is followed by
76919 `=word`, then the value of that variable shall be set to `word`.``. The reason for this change is that the
76920 SYNOPSIS for `export` includes:

```
76921 export name [=word] . . .
```

76922 but the meaning of the optional `=word` is never explained in the text.

76923 **Issue 7**

76924 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0043 [352] is applied.

76925 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0051 [654] and XCU/TC2-2008/0052
76926 [960] are applied.

76927 **NAME**

76928 readonly — set the readonly attribute for variables

76929 **SYNOPSIS**76930 readonly name[=*word*] . . .

76931 readonly -p

76932 **DESCRIPTION**

76933 The variables whose *names* are specified shall be given the *readonly* attribute. The values of
 76934 variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those
 76935 variables be unset by the *unset* utility. If the name of a variable is followed by =*word*, then the
 76936 value of that variable shall be set to *word*.

76937 The *readonly* special built-in shall support XBD [Section 12.2](#).

76938 When -p is specified, *readonly* writes to the standard output the names and values of all read-
 76939 only variables, in the following format:

76940 "readonly %s=%s\n", <name>, <value>

76941 if *name* is set, and

76942 "readonly %s\n", <name>

76943 if *name* is unset.

76944 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 76945 reinput to the shell as commands that achieve the same value and *readonly* attribute-setting
 76946 results in a shell execution environment in which:

- 76947 1. Variables with values at the time they were output do not have the *readonly* attribute set.
- 76948 2. Variables that were unset at the time they were output do not have a value at the time at
 76949 which the saved output is reinput to the shell.

76950 When no arguments are given, the results are unspecified.

76951 **OPTIONS**

76952 See the DESCRIPTION.

76953 **OPERANDS**

76954 See the DESCRIPTION.

76955 **STDIN**

76956 Not used.

76957 **INPUT FILES**

76958 None.

76959 **ENVIRONMENT VARIABLES**

76960 None.

76961 **ASYNCHRONOUS EVENTS**

76962 Default.

76963 **STDOUT**

76964 See the DESCRIPTION.

76965 **STDERR**

76966 The standard error shall be used only for diagnostic messages.

76967 **OUTPUT FILES**

76968 None.

76969 **EXTENDED DESCRIPTION**

76970 None.

76971 **EXIT STATUS**76972 0 All *name* operands were successfully marked readonly.76973 >0 At least one *name* could not be marked readonly, or the `-p` option was specified and an error
76974 occurred.76975 **CONSEQUENCES OF ERRORS**

76976 Default.

76977 **APPLICATION USAGE**

76978 None.

76979 **EXAMPLES**76980 `readonly HOME PWD`76981 **RATIONALE**76982 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of
76983 POSIX.1-2017 allows this behavior, but does not require it.76984 The `-p` option allows portable access to the values that can be saved and then later restored
76985 using, for example, a *dot* script. Also see the RATIONALE for *export* for a description of the no-
76986 argument and `-p` output cases and a related example.76987 Read-only functions were considered, but they were omitted as not being historical practice or
76988 particularly useful. Furthermore, functions must not be read-only across invocations to preclude
76989 “spoofing” (spoofing is the term for the practice of creating a program that acts like a well-
76990 known utility with the intent of subverting the real intent of the user) of administrative or
76991 security-relevant (or security-conscious) shell scripts.76992 **FUTURE DIRECTIONS**

76993 None.

76994 **SEE ALSO**76995 [Section 2.14](#)76996 [XBD Section 12.2](#)76997 **CHANGE HISTORY**76998 **Issue 6**

76999 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

77000 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
77001 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
77002 behavior is intended.77003 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to
77004 the end of the first paragraph of the DESCRIPTION: “If the name of a variable is followed by
77005 `=word`, then the value of that variable shall be set to *word*.”. The reason for this change is that the
77006 SYNOPSIS for *readonly* includes:

77007 `readonly name[=word]` . . .

77008 but the meaning of the optional ``=*word*'' is never explained in the text.

77009 **Issue 7**

77010 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0052 [960] is applied.

77011 **NAME**

77012 return — return from a function or dot script

77013 **SYNOPSIS**77014 return [*n*]77015 **DESCRIPTION**77016 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the
77017 shell is not currently executing a function or *dot* script, the results are unspecified.77018 **OPTIONS**

77019 None.

77020 **OPERANDS**

77021 See the DESCRIPTION.

77022 **STDIN**

77023 Not used.

77024 **INPUT FILES**

77025 None.

77026 **ENVIRONMENT VARIABLES**

77027 None.

77028 **ASYNCHRONOUS EVENTS**

77029 Default.

77030 **STDOUT**

77031 Not used.

77032 **STDERR**

77033 The standard error shall be used only for diagnostic messages.

77034 **OUTPUT FILES**

77035 None.

77036 **EXTENDED DESCRIPTION**

77037 None.

77038 **EXIT STATUS**77039 The value of the special parameter '?' shall be set to *n*, an unsigned decimal integer, or to the
77040 exit status of the last command executed if *n* is not specified. If *n* is not an unsigned decimal
77041 integer, or is greater than 255, the results are unspecified. When *return* is executed in a *trap*
77042 action, the last command is considered to be the command that executed immediately preceding
77043 the *trap* action.77044 **CONSEQUENCES OF ERRORS**

77045 Default.

77046 **APPLICATION USAGE**

77047 None.

77048 **EXAMPLES**

77049 None.

77050 **RATIONALE**77051 The behavior of *return* when not in a function or *dot* script differs between the System V shell
77052 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is
77053 the same as *exit*.

77054 The results of returning a number greater than 255 are undefined because of differing practices
77055 in the various historical implementations. Some shells AND out all but the low-order 8 bits;
77056 others allow larger values, but not of unlimited size.

77057 See the discussion of appropriate exit status values under *exit*.

77058 **FUTURE DIRECTIONS**

77059 None.

77060 **SEE ALSO**

77061 [Section 2.9.5](#), [Section 2.14](#), *dot*

77062 **CHANGE HISTORY**

77063 **Issue 6**

77064 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
77065 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
77066 behavior is intended.

77067 **Issue 7**

77068 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0044 [214] and XCU/TC1-2008/0045
77069 [214] are applied.

77070 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0052 [960] is applied.

77071 **NAME**77072 set `[-o option] [argument...]`77073 **SYNOPSIS**77074 set `[-abCefhmnvux] [-o option] [argument...]`77075 set `[+abCefhmnvux] [+o option] [argument...]`77076 set `-- [argument...]`77077 set `-o`77078 set `+o`77079 **DESCRIPTION**77080 If no *options* or *arguments* are specified, *set* shall write the names and values of all shell variables
77081 in the collation sequence of the current locale. Each *name* shall start on a separate line, using the
77082 format:

77083 "%s=%s\n", <name>, <value>

77084 The *value* string shall be written with appropriate quoting; see the description of shell quoting in
77085 [Section 2.2](#). The output shall be suitable for reinput to the shell, setting or resetting, as far as
77086 possible, the variables that are currently set; read-only variables cannot be reset.77087 When options are specified, they shall set or unset attributes of the shell, as described below.
77088 When *arguments* are specified, they cause positional parameters to be set or unset, as described
77089 below. Setting or unsetting attributes and positional parameters are not necessarily related
77090 actions, but they can be combined in a single invocation of *set*.77091 The *set* special built-in shall support XBD [Section 12.2](#) except that options can be specified with
77092 either a leading <hyphen-minus> (meaning enable the option) or <plus-sign> (meaning disable
77093 it) unless otherwise specified.77094 Implementations shall support the options in the following list in both their <hyphen-minus>
77095 and <plus-sign> forms. These options can also be specified as options to *sh*.77096 **-a** When this option is on, the *export* attribute shall be set for each variable to which an
77097 assignment is performed; see XBD [Section 4.23](#). If the assignment precedes a utility name in
77098 a command, the *export* attribute shall not persist in the current execution environment after
77099 the utility completes, with the exception that preceding one of the special built-in utilities
77100 causes the *export* attribute to persist after the built-in has completed. If the assignment does
77101 not precede a utility name in the command, or if the assignment is a result of the operation
77102 of the *getopts* or *read* utilities, the *export* attribute shall persist until the variable is unset.77103 **-b** This option shall be supported if the implementation supports the User Portability Utilities
77104 option. It shall cause the shell to notify the user asynchronously of background job
77105 completions. The following message is written to standard error:

77106 "[%d]%c %s%s\n", <job-number>, <current>, <status>, <job-name>

77107 where the fields shall be as follows:

77108 <current> The character '+' identifies the job that would be used as a default for
77109 the *fg* or *bg* utilities; this job can also be specified using the *job_id* "%+" or
77110 "%%". The character '-' identifies the job that would become the default
77111 if the current default job were to exit; this job can also be specified using
77112 the *job_id* "%-". For other jobs, this field is a <space>. At most one job
77113 can be identified with '+' and at most one job can be identified with '-'.
77114 If there is any suspended job, then the current job shall be a suspended

- 77115 job. If there are at least two suspended jobs, then the previous job also
77116 shall be a suspended job.
- 77117 `<job-number>` A number that can be used to identify the process group to the *wait*, *fg*, *bg*,
77118 and *kill* utilities. Using these utilities, the job can be identified by prefixing
77119 the job number with '`%`'.
- 77120 `<status>` Unspecified.
- 77121 `<job-name>` Unspecified.
- 77122 When the shell notifies the user a job has been completed, it may remove the job's process
77123 ID from the list of those known in the current shell execution environment; see [Section](#)
77124 [2.9.3.1](#). Asynchronous notification shall not be enabled by default.
- 77125 **-C** (Uppercase C.) Prevent existing files from being overwritten by the shell's '`>`' redirection
77126 operator (see [Section 2.7.2](#)); the '`>|`' redirection operator shall override this *noclobber*
77127 option for an individual file.
- 77128 **-e** When this option is on, when any command fails (for any of the reasons listed in [Section](#)
77129 [2.8.1](#) or by returning an exit status greater than zero), the shell immediately shall exit, as if
77130 by executing the *exit* special built-in utility with no arguments, with the following
77131 exceptions:
- 77132 1. The failure of any individual command in a multi-command pipeline shall not cause
77133 the shell to exit. Only the failure of the pipeline itself shall be considered.
 - 77134 2. The **-e** setting shall be ignored when executing the compound list following the
77135 **while**, **until**, **if**, or **elif** reserved word, a pipeline beginning with the **!** reserved word,
77136 or any command of an AND-OR list other than the last.
 - 77137 3. If the exit status of a compound command other than a subshell command was the
77138 result of a failure while **-e** was being ignored, then **-e** shall not apply to this
77139 command.
- 77140 This requirement applies to the shell environment and each subshell environment
77141 separately. For example, in:
- ```
77142 set -e; (false; echo one) | cat; echo two
```
- 77143 the *false* command causes the subshell to exit without executing *echo one*; however, *echo*  
77144 *two* is executed because the exit status of the pipeline `(false; echo one) | cat` is  
77145 zero.
- 77146 **-f** The shell shall disable pathname expansion.
- 77147 **-h** Locate and remember utilities invoked by functions as those functions are defined (the  
77148 utilities are normally located when the function is executed).
- 77149 **-m** This option shall be supported if the implementation supports the User Portability Utilities  
77150 option. All jobs shall be run in their own process groups. Immediately before the shell issues  
77151 a prompt after completion of the background job, a message reporting the exit status of the  
77152 background job shall be written to standard error. If a foreground job stops, the shell shall  
77153 write a message to standard error to that effect, formatted as described by the *jobs* utility. In  
77154 addition, if a job changes status other than exiting (for example, if it stops for input or  
77155 output or is stopped by a SIGSTOP signal), the shell shall write a similar message  
77156 immediately prior to writing the next prompt. This option is enabled by default for  
77157 interactive shells.

- 77158        **-n** The shell shall read commands but does not execute them; this can be used to check for  
77159        shell script syntax errors. An interactive shell may ignore this option.
- 77160        **-o** Write the current settings of the options to standard output in an unspecified format.
- 77161        **+o** Write the current option settings to standard output in a format that is suitable for reinput  
77162        to the shell as commands that achieve the same options settings.
- 77163        **-o option**
- 77164        This option is supported if the system supports the User Portability Utilities option. It shall  
77165        set various options, many of which shall be equivalent to the single option letters. The  
77166        following values of *option* shall be supported:
- 77167        *allexport*       Equivalent to **-a**.
- 77168        *errexit*        Equivalent to **-e**.
- 77169        *ignoreeof*       Prevent an interactive shell from exiting on end-of-file. This setting prevents  
77170        accidental logouts when <control>-D is entered. A user shall explicitly *exit* to  
77171        leave the interactive shell.
- 77172        *monitor*        Equivalent to **-m**. This option is supported if the system supports the User  
77173        Portability Utilities option.
- 77174        *noclobber*       Equivalent to **-C** (uppercase C).
- 77175        *noglob*         Equivalent to **-f**.
- 77176        *noexec*         Equivalent to **-n**.
- 77177        *nolog*         Prevent the entry of function definitions into the command history; see  
77178        [Command History List](#).
- 77179        *notify*         Equivalent to **-b**.
- 77180        *nounset*        Equivalent to **-u**.
- 77181        *verbose*        Equivalent to **-v**.
- 77182        *vi*            Allow shell command line editing using the built-in *vi* editor. Enabling *vi*  
77183        mode shall disable any other command line editing mode provided as an  
77184        implementation extension.
- 77185        It need not be possible to set *vi* mode on for certain block-mode terminals.
- 77186        *xtrace*         Equivalent to **-x**.
- 77187        **-u** When the shell tries to expand an unset parameter other than the '@' and '\*' special  
77188        parameters, it shall write a message to standard error and the expansion shall fail with the  
77189        consequences specified in [Section 2.8.1](#).
- 77190        **-v** The shell shall write its input to standard error as it is read.
- 77191        **-x** The shell shall write to standard error a trace for each command after it expands the  
77192        command and before it executes it. It is unspecified whether the command that turns  
77193        tracing off is traced.
- 77194        The default for all these options shall be off (unset) unless stated otherwise in the description of  
77195        the option or unless the shell was invoked with them on; see *sh*.
- 77196        The remaining arguments shall be assigned in order to the positional parameters. The special  
77197        parameter '#' shall be set to reflect the number of positional parameters. All positional  
77198        parameters shall be unset before any new values are assigned.

77199 If the first argument is '-', the results are unspecified.

77200 The special argument "--" immediately following the *set* command name can be used to  
77201 delimit the arguments if the first argument begins with '+' or '-', or to prevent inadvertent  
77202 listing of all shell variables when there are no arguments. The command *set --* without *argument*  
77203 shall unset all positional parameters and set the special parameter '#' to zero.

**77204 OPTIONS**

77205 See the DESCRIPTION.

**77206 OPERANDS**

77207 See the DESCRIPTION.

**77208 STDIN**

77209 Not used.

**77210 INPUT FILES**

77211 None.

**77212 ENVIRONMENT VARIABLES**

77213 None.

**77214 ASYNCHRONOUS EVENTS**

77215 Default.

**77216 STDOUT**

77217 See the DESCRIPTION.

**77218 STDERR**

77219 The standard error shall be used only for diagnostic messages.

**77220 OUTPUT FILES**

77221 None.

**77222 EXTENDED DESCRIPTION**

77223 None.

**77224 EXIT STATUS**

77225 0 Successful completion.

77226 >0 An invalid option was specified, or an error occurred.

**77227 CONSEQUENCES OF ERRORS**

77228 Default.

**77229 APPLICATION USAGE**

77230 Application writers should avoid relying on *set -e* within functions. For example, in the  
77231 following script:

```
77232 set -e
77233 start() {
77234 some_server
77235 echo some_server started successfully
77236 }
77237 start || echo >&2 some_server failed
```

77238 the *-e* setting is ignored within the function body (because the function is a command in an  
77239 AND-OR list other than the last). Therefore, if *some\_server* fails, the function carries on to  
77240 echo "some\_server started successfully", and the exit status of the function is zero  
77241 (which means "some\_server failed" is not output).

77242 **EXAMPLES**

77243 Write out all variables and their values:

77244 `set`

77245 Set \$1, \$2, and \$3 and set "\$#" to 3:

77246 `set c a b`77247 Turn on the `-x` and `-v` options:77248 `set -xv`

77249 Unset all positional parameters:

77250 `set --`77251 Set \$1 to the value of `x`, even if it begins with '-' or '+':77252 `set -- "$x"`77253 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading '-' or '+':77254 `set -- $x`77255 **RATIONALE**

77256 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the  
 77257 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether  
 77258 the `set --` form might be misinterpreted as being equivalent to `set` without any options or  
 77259 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`  
 77260 `--` only unsets parameters if there is at least one argument; the only way to unset all parameters  
 77261 is to use *shift*. Using the KornShell version should not affect System V scripts because there  
 77262 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

77263 `set -- "$@"`

77264 and there were in fact no arguments resulting from `"$@"`, unsetting the parameters would have  
 77265 no result.

77266 The `set +` form in early proposals was omitted as being an unnecessary duplication of `set` alone  
 77267 and not widespread historical practice.

77268 The *noclobber* option was changed to allow `set -C` as well as the `set -o noclobber` option. The  
 77269 single-letter version was added so that the historical "\$-" paradigm would not be broken; see  
 77270 [Section 2.5.2](#).

77271 The description of the `-e` option is intended to match the behavior of the 1988 version of the  
 77272 KornShell.

77273 The `-h` flag is related to command name hashing. See *hash*.

77274 The following `set` flags were omitted intentionally with the following rationale:

77275 **-k** The `-k` flag was originally added by the author of the Bourne shell to make it easier for  
 77276 users of pre-release versions of the shell. In early versions of the Bourne shell the construct  
 77277 `set name=value` had to be used to assign values to shell variables. The problem with `-k` is  
 77278 that the behavior affects parsing, virtually precluding writing any compilers. To explain the  
 77279 behavior of `-k`, it is necessary to describe the parsing algorithm, which is implementation-  
 77280 defined. For example:

77281 `set -k; echo name=value`

77282 and:

```
77283 set -k
77284 echo name=value
```

77285 behave differently. The interaction with functions is even more complex. What is more, the  
77286 `-k` flag is never needed, since the command line could have been reordered.

77287 `-t` The `-t` flag is hard to specify and almost never used. The only known use could be done  
77288 with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page  
77289 says that it exits after reading and executing one command. What is one command? If the  
77290 input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

77291 Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion  
77292 was that the *unset* utility should be used to unset options instead of using the non-*getopt*(-)-able  
77293 `+option` syntax. However, the conclusion was reached that the historical practice of using `+option`  
77294 was satisfactory and that there was no compelling reason to modify such widespread historical  
77295 practice.

77296 The `-o` option was adopted from the KornShell to address user needs. In addition to its  
77297 generally friendly interface, `-o` is needed to provide the *vi* command line editing mode, for  
77298 which historical practice yields no single-letter option name. (Although it might have been  
77299 possible to invent such a letter, it was recognized that other editing modes would be developed  
77300 and `-o` provides ample name space for describing such extensions.)

77301 Historical implementations are inconsistent in the format used for `-o` option status reporting.  
77302 The `+o` format without an option-argument was added to allow portable access to the options  
77303 that can be saved and then later restored using, for instance, a dot script.

77304 Historically, *sh* did trace the command *set +x*, but *ksh* did not.

77305 The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically  
77306 `<control>-D`) is entered. A user shall explicitly *exit* to leave the interactive shell.

77307 The *set -m* option was added to apply only to the UPE because it applies primarily to interactive  
77308 use, not shell script applications.

77309 The ability to do asynchronous notification became available in the 1988 version of the  
77310 KornShell. To have it occur, the user had to issue the command:

```
77311 trap "jobs -n" CLD
```

77312 The C shell provides two different levels of an asynchronous notification capability. The  
77313 environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When set, it  
77314 notifies the user immediately of background job completions. When unset, this capability is  
77315 turned off.

77316 The other notification ability comes through the built-in utility *notify*. The syntax is:

```
77317 notify [%job ...]
```

77318 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when  
77319 the state of the current job changes. If given operands, *notify* asynchronously informs the user of  
77320 changes in the states of the specified jobs.

77321 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,  
77322 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX  
77323 environment variable name).

77324 The *set -b* option was selected as a compromise.

77325 The *notify* built-in was considered to have more functionality than was required for simple

- 77326 asynchronous notification.
- 77327 Historically, some shells applied the `-u` option to all parameters including `$@` and `$*`. The  
77328 standard developers felt that this was a misfeature since it is normal and common for `$@` and `$*`  
77329 to be used in shell scripts regardless of whether they were passed any arguments. Treating these  
77330 uses as an error when no arguments are passed reduces the value of `-u` for its intended purpose  
77331 of finding spelling mistakes in variable names and uses of unset positional parameters.
- 77332 **FUTURE DIRECTIONS**
- 77333 None.
- 77334 **SEE ALSO**
- 77335 [Section 2.14, \*hash\*](#)
- 77336 [XBD Section 4.23, Section 12.2](#)
- 77337 **CHANGE HISTORY**
- 77338 **Issue 6**
- 77339 The obsolescent *set* command name followed by `'-'` has been removed.
- 77340 The following new requirements on POSIX implementations derive from alignment with the  
77341 Single UNIX Specification:
- 77342 The *nolog* option is added to *set -o*.
- 77343 IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes  
77344 into account the description of the option.
- 77345 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
77346 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
77347 behavior is intended.
- 77348 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square  
77349 brackets in the example in RATIONALE to be in bold, which is the typeface used for optional  
77350 items.
- 77351 **Issue 7**
- 77352 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first  
77353 argument is `'-'`.
- 77354 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 77355 XSI shading is removed from the `-h` functionality.
- 77356 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0046 [52], XCU/TC1-2008/0047  
77357 [155,280], XCU/TC1-2008/0048 [52], XCU/TC1-2008/0049 [52], and XCU/TC1-2008/0050  
77358 [155,430] are applied.
- 77359 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0053 [584], XCU/TC2-2008/0054  
77360 [717], XCU/TC2-2008/0055 [717], and XCU/TC2-2008/0056 [960] are applied.

**77361 NAME**

77362 shift ‡'shift positional parameters

**77363 SYNOPSIS**

77364 shift [*n*]

**77365 DESCRIPTION**

77366 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of  
77367 parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The  
77368 parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the  
77369 parameter '#' is updated to reflect the new number of positional parameters.

77370 The value *n* shall be an unsigned decimal integer less than or equal to the value of the special  
77371 parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special  
77372 parameters are not changed.

**77373 OPTIONS**

77374 None.

**77375 OPERANDS**

77376 See the DESCRIPTION.

**77377 STDIN**

77378 Not used.

**77379 INPUT FILES**

77380 None.

**77381 ENVIRONMENT VARIABLES**

77382 None.

**77383 ASYNCHRONOUS EVENTS**

77384 Default.

**77385 STDOUT**

77386 Not used.

**77387 STDERR**

77388 The standard error shall be used only for diagnostic messages.

**77389 OUTPUT FILES**

77390 None.

**77391 EXTENDED DESCRIPTION**

77392 None.

**77393 EXIT STATUS**

77394 If the *n* operand is invalid or is greater than "\$#", this may be considered a syntax error and a  
77395 non-interactive shell may exit; if the shell does not exit in this case, a non-zero exit status shall be  
77396 returned. Otherwise, zero shall be returned.

**77397 CONSEQUENCES OF ERRORS**

77398 Default.

77399 **APPLICATION USAGE**

77400 None.

77401 **EXAMPLES**77402 `$ set a b c d e`77403 `$ shift 2`77404 `$ echo $*`77405 `c d e`77406 **RATIONALE**

77407 None.

77408 **FUTURE DIRECTIONS**

77409 None.

77410 **SEE ALSO**77411 [Section 2.14](#)77412 **CHANGE HISTORY**77413 **Issue 6**

77414 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
77415 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
77416 behavior is intended.

77417 **Issue 7**

77418 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0051 [459] is applied.



77419 **NAME**

77420 times — write process times

77421 **SYNOPSIS**

77422 times

77423 **DESCRIPTION**77424 The *times* utility shall write the accumulated user and system times for the shell and for all of its  
77425 child processes, in the following POSIX locale format:77426 "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,  
77427 <shell user seconds>, <shell system minutes>,  
77428 <shell system seconds>, <children user minutes>,  
77429 <children user seconds>, <children system minutes>,  
77430 <children system seconds>77431 The four pairs of times shall correspond to the members of the <sys/times.h> **tms** structure  
77432 (defined in XBD [Chapter 13](#)) as returned by *times()*: *tms\_utime*, *tms\_stime*, *tms\_cutime*, and  
77433 *tms\_cstime*, respectively.77434 **OPTIONS**

77435 None.

77436 **OPERANDS**

77437 None.

77438 **STDIN**

77439 Not used.

77440 **INPUT FILES**

77441 None.

77442 **ENVIRONMENT VARIABLES**

77443 None.

77444 **ASYNCHRONOUS EVENTS**

77445 Default.

77446 **STDOUT**

77447 See the DESCRIPTION.

77448 **STDERR**

77449 The standard error shall be used only for diagnostic messages.

77450 **OUTPUT FILES**

77451 None.

77452 **EXTENDED DESCRIPTION**

77453 None.

77454 **EXIT STATUS**

77455 0 Successful completion.

77456 &gt;0 An error occurred.

77457 **CONSEQUENCES OF ERRORS**

77458 Default.

77459 **APPLICATION USAGE**

77460 None.

77461 **EXAMPLES**77462 `$ times`77463 `0m0.43s 0m1.11s`77464 `8m44.18s 1m43.23s`77465 **RATIONALE**77466 The *times* special built-in from the Single UNIX Specification is now required for all conforming  
77467 shells.77468 **FUTURE DIRECTIONS**

77469 None.

77470 **SEE ALSO**77471 [Section 2.14](#)77472 XBD [<sys/times.h>](#)77473 **CHANGE HISTORY**77474 **Issue 6**77475 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the  
77476 DESCRIPTION from: "Write the accumulated user and system times for the shell and for all of  
77477 its child processes ..." to: "The *times* utility shall write the accumulated user and system times for  
77478 the shell and for all of its child processes ...".77479 **Issue 7**

77480 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0056 [960] is applied.

77481 **NAME**

77482 trap ‡trap signals

77483 **SYNOPSIS**77484 trap *n* [*condition...*]77485 trap [*action condition...*]77486 **DESCRIPTION**

77487 If the first operand is an unsigned decimal integer, the shell shall treat all operands as  
 77488 conditions, and shall reset each condition to the default value. Otherwise, if there are operands,  
 77489 the first is treated as an action and the remaining as conditions.

77490 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (""), the  
 77491 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read  
 77492 and executed by the shell when one of the corresponding conditions arises. The action of *trap*  
 77493 shall override a previous action (either default action or one explicitly set). The value of "\$?"  
 77494 after the *trap* action completes shall be the value it had before *trap* was invoked.

77495 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,  
 77496 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in  
 77497 XBD Chapter 13; for example, HUP, INT, QUIT, TERM. Implementations may permit names  
 77498 with the SIG prefix or ignore case in signal names as an extension. Setting a trap for SIGKILL or  
 77499 SIGSTOP produces undefined results.

77500 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment  
 77501 immediately after the last command executed before the *trap* on EXIT was taken.

77502 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

77503 eval *action*

77504 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although  
 77505 no error need be reported when attempting to do so. An interactive shell may reset or catch  
 77506 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed  
 77507 with another *trap* command.

77508 When a subshell is entered, traps that are not being ignored shall be set to the default actions,  
 77509 except in the case of a command substitution containing only a single *trap* command, when the  
 77510 traps need not be altered. Implementations may check for this case using only lexical analysis;  
 77511 for example, if `trap` and  $\$( trap -- )$  do not alter the traps in the subshell, cases such as  
 77512 assigning `var=trap` and then using  $\$(\$var)$  may still alter them. This does not imply that the  
 77513 *trap* command cannot be used within the subshell to set new traps.

77514 The *trap* command with no operands shall write to standard output a list of commands  
 77515 associated with each condition. If the command is executed in a subshell, the implementation  
 77516 does not perform the optional check described above for a command substitution containing  
 77517 only a single *trap* command, and no *trap* commands with operands have been executed since  
 77518 entry to the subshell, the list shall contain the commands that were associated with each  
 77519 condition immediately before the subshell environment was entered. Otherwise, the list shall  
 77520 contain the commands currently associated with each condition. The format shall be:

77521 "trap -- %s %s ... \n", &lt;action&gt;, &lt;condition&gt; ...

77522 The shell shall format the output, including the proper use of quoting, so that it is suitable for  
 77523 reinput to the shell as commands that achieve the same trapping results. For example:

77524 save\_traps=\$(trap)

77525 ...

77526 eval "\$save\_traps"

77527 XSI XSI-conformant systems also allow numeric signal numbers for the conditions corresponding to  
 77528 the following signal names:

|       |    |         |
|-------|----|---------|
| 77529 | 1  | SIGHUP  |
| 77530 | 2  | SIGINT  |
| 77531 | 3  | SIGQUIT |
| 77532 | 6  | SIGABRT |
| 77533 | 9  | SIGKILL |
| 77534 | 14 | SIGALRM |
| 77535 | 15 | SIGTERM |

77536 The *trap* special built-in shall conform to XBD [Section 12.2](#).

77537 **OPTIONS**

77538 None.

77539 **OPERANDS**

77540 See the DESCRIPTION.

77541 **STDIN**

77542 Not used.

77543 **INPUT FILES**

77544 None.

77545 **ENVIRONMENT VARIABLES**

77546 None.

77547 **ASYNCHRONOUS EVENTS**

77548 Default.

77549 **STDOUT**

77550 See the DESCRIPTION.

77551 **STDERR**

77552 The standard error shall be used only for diagnostic messages.

77553 **OUTPUT FILES**

77554 None.

77555 **EXTENDED DESCRIPTION**

77556 None.

77557 **EXIT STATUS**

77558 XSI If the trap name **or number** is invalid, a non-zero exit status shall be returned; otherwise, zero  
 77559 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names **or**  
 77560 **numbers** shall not be considered a syntax error and do not cause the shell to abort.

77561 **CONSEQUENCES OF ERRORS**

77562 Default.

77563 **APPLICATION USAGE**

77564 None.

77565 **EXAMPLES**

77566 Write out a list of all traps and actions:

77567 trap

77568 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable  
77569 executes when the shell terminates:

77570 trap '"\$HOME"/logout' EXIT

77571 or:

77572 trap '"\$HOME"/logout' 0

77573 Unset traps on INT, QUIT, TERM, and EXIT:

77574 trap - INT QUIT TERM EXIT

77575 **RATIONALE**77576 Implementations may permit lowercase signal names as an extension. Implementations may  
77577 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*  
77578 utilities in this volume of POSIX.1-2017 are now consistent in their omission of the SIG prefix for  
77579 signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the signals  
77580 without prefixes.77581 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but  
77582 it has no effect. Portable POSIX applications cannot attempt to trap these signals.77583 The output format is not historical practice. Since the output of historical *trap* commands is not  
77584 portable (because numeric signal values are not portable) and had to change to become so, an  
77585 opportunity was taken to format the output in a way that a shell script could use to save and  
77586 then later reuse a trap if it wanted.77587 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is  
77588 allowable as an extension, but was not mandated, as other shells have not used it.77589 The text about the environment for the EXIT trap invalidates the behavior of some historical  
77590 versions of interactive shells which, for example, close the standard input before executing a  
77591 trap on 0. For example, in some historical interactive shell sessions the following trap on 0  
77592 would always print "--":

77593 trap 'read foo; echo "--\$foo--"' 0

77594 The command:

77595 trap 'eval " \$cmd"' 0

77596 causes the contents of the shell variable *cmd* to be executed as a command when the shell exits.  
77597 Using:

77598 trap '\$cmd' 0

77599 does not work correctly if *cmd* contains any special characters such as quoting or redirections.  
77600 Using:

77601 trap " \$cmd" 0

77602 also works (the leading <space> character protects against unlikely cases where *cmd* is a decimal  
77603 integer or begins with '-'), but it expands the *cmd* variable when the *trap* command is executed,

77604 not when the exit action is executed.

77605 **FUTURE DIRECTIONS**

77606 None.

77607 **SEE ALSO**

77608 [Section 2.14](#)

77609 XBD [Section 12.2, <signal.h>](#)

77610 **CHANGE HISTORY**

77611 **Issue 6**

77612 XSI-conforming implementations provide the mapping of signal names to numbers given above  
77613 (previously this had been marked obsolescent). Other implementations need not provide this  
77614 optional mapping.

77615 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
77616 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
77617 behavior is intended.

77618 **Issue 7**

77619 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

77620 Austin Group Interpretation 1003.1-2001 #116 is applied.

77621 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0052 [53,268,440],  
77622 XCU/TC1-2008/0053 [53,268,440], XCU/TC1-2008/0054 [163], XCU/TC1-2008/0055 [163], and  
77623 XCU/TC1-2008/0056 [163] are applied.

77624 **NAME**77625           unset     $\ddagger$ 'unset values and attributes of variables and functions77626 **SYNOPSIS**77627           unset [-fv] *name*...77628 **DESCRIPTION**77629           Each variable or function specified by *name* shall be unset.77630           If **-v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from  
77631           the environment. Read-only variables cannot be unset.77632           If **-f** is specified, *name* refers to a function and the shell shall unset the function definition.77633           If neither **-f** nor **-v** is specified, *name* refers to a variable; if a variable by that name does not  
77634           exist, it is unspecified whether a function by that name, if any, shall be unset.77635           Unsetting a variable or function that was not previously set shall not be considered an error and  
77636           does not cause the shell to abort.77637           The *unset* special built-in shall support XBD [Section 12.2](#).

77638           Note that:

77639           VARIABLE=

77640           is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to " ". Also, the  
77641           variables that can be *unset* should not be misinterpreted to include the special parameters (see  
77642           [Section 2.5.2](#)).77643 **OPTIONS**

77644           See the DESCRIPTION.

77645 **OPERANDS**

77646           See the DESCRIPTION.

77647 **STDIN**

77648           Not used.

77649 **INPUT FILES**

77650           None.

77651 **ENVIRONMENT VARIABLES**

77652           None.

77653 **ASYNCHRONOUS EVENTS**

77654           Default.

77655 **STDOUT**

77656           Not used.

77657 **STDERR**

77658           The standard error shall be used only for diagnostic messages.

77659 **OUTPUT FILES**

77660           None.

77661 **EXTENDED DESCRIPTION**

77662           None.

77663 **EXIT STATUS**77664           0 All *name* operands were successfully unset.77665           >0 At least one *name* could not be unset.77666 **CONSEQUENCES OF ERRORS**

77667           Default.

77668 **APPLICATION USAGE**

77669           None.

77670 **EXAMPLES**77671           Unset *VISUAL* variable:

77672           unset -v VISUAL

77673           Unset the functions **foo** and **bar**:

77674           unset -f foo bar

77675 **RATIONALE**77676           Consideration was given to omitting the **-f** option in favor of an *unfunction* utility, but the  
77677           standard developers decided to retain historical practice.77678           The **-v** option was introduced because System V historically used one name space for both  
77679           variables and functions. When *unset* is used without options, System V historically unset either a  
77680           function or a variable, and there was no confusion about which one was intended. A portable  
77681           POSIX application can use *unset* without an option to unset a variable, but not a function; the **-f**  
77682           option must be used.77683 **FUTURE DIRECTIONS**

77684           None.

77685 **SEE ALSO**77686           [Section 2.14](#)77687           XBD [Section 12.2](#)77688 **CHANGE HISTORY**77689 **Issue 6**77690           IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page  
77691           sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in  
77692           behavior is intended.77693 **Issue 7**

77694           SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.





# Batch Environment Services

77697 OB BE This chapter describes the services and utilities that shall be implemented on all systems that  
 77698 claim conformance to the Batch Environment Services and Utilities option. The functionality  
 77699 described in this section shall be provided on implementations that support the Batch  
 77700 Environment Services and Utilities option (and the rest of this section is not further shaded for  
 77701 this option).

77702 Note that the Batch Environment Services and Utilities option is marked obsolescent in Issue 7.

## 3.1 General Concepts

### 3.1.1 Batch Client-Server Interaction

77705 Batch jobs are created and managed by batch servers. A batch client interacts with a batch server  
 77706 to access batch services on behalf of the user. In order to use batch services, a user must have  
 77707 access to a batch client.

77708 A batch server is a computational entity, such as a daemon process, that provides batch services.  
 77709 Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

77710 The batch utilities described in this volume of POSIX.1-2017 (and listed in [Table 3-1](#)) are clients  
 77711 of batch services; they allow users to perform actions on the job such as creating, modifying, and  
 77712 deleting batch jobs from a shell command line. Although these batch utilities may be said to  
 77713 accomplish certain services, they actually obtain services on behalf of a user by means of  
 77714 requests to batch servers.

**Table 3-1** Batch Utilities

|       |               |               |                |              |
|-------|---------------|---------------|----------------|--------------|
| 77716 | <i>qalter</i> | <i>qmove</i>  | <i>qrls</i>    | <i>qstat</i> |
| 77717 | <i>qdel</i>   | <i>qmsg</i>   | <i>qselect</i> | <i>qsub</i>  |
| 77718 | <i>qhold</i>  | <i>qrerun</i> | <i>qsig</i>    |              |

77719 Client-server interaction takes place by means of the batch requests defined in this chapter.  
 77720 Because direct access to batch jobs and queues is limited to batch servers, clients and servers of  
 77721 different implementations can interoperate, since dependencies on private structures for batch  
 77722 jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch  
 77723 servers.

### 77724 3.1.2 Batch Queues

77725 Two types of batch queue are described: routing queues and execution queues. When a batch job  
77726 is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing  
77727 queues under the following conditions:

77728         The batch job has been routed to another queue.

77729         The batch job has been deleted from the batch queue.

77730         The batch job has been aborted.

77731 When a batch job is placed in an execution queue, it is a candidate for execution.

77732 A batch job is removed from an execution queue under the following conditions:

77733         The batch job has been executed and exited.

77734         The batch job has been aborted.

77735         The batch job has been deleted from the batch queue.

77736         The batch job has been moved to another queue.

77737 Access to a batch queue is limited to the batch server that manages the batch queue. Clients  
77738 never access a batch queue or a batch job directly, either to read or write information; all client  
77739 access to batch queues or jobs takes place through batch servers.

### 77740 3.1.3 Batch Job Creation

77741 When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier  
77742 to the job. A batch job identifier consists of both a sequence number that is unique among the  
77743 sequence numbers issued by that server and the name of the server. Since the batch server name  
77744 is unique within a name space, the job identifier is likewise unique within the name space.

77745 The batch server that creates a batch job shall return the batch server-assigned job identifier to  
77746 the client that requested the job creation. If the batch server routes or moves the job to another  
77747 server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job  
77748 shall never change.

### 77749 3.1.4 Batch Job Tracking

77750 Since a batch job may be moved after creation, the batch server name component of the job  
77751 identifier need not indicate the location of the job. An implementation may provide a batch job  
77752 tracking mechanism, in which case the user generally does not need to know the location of the  
77753 job. However, an implementation need not provide a batch job tracking mechanism, in which  
77754 case the user must find routed jobs by probing the possible destinations.

### 77755 3.1.5 Batch Job Routing

77756 To route a batch job, a batch server either moves the job to some other queue that is managed by  
77757 the batch server, or requests that some other batch server accept the job.

77758 Each routing queue has one or more queues to which it can route batch jobs. The batch server  
77759 administrator creates routing queues.

77760 A batch server may route a batch job from a routing queue to another routing queue. Batch  
77761 servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a  
77762 batch server routes jobs from the routing queues that it manages. The algorithm by which a  
77763 batch server selects a batch queue to which to route a batch job is implementation-defined.

77764 A batch job need not be eligible for routing to all the batch queues fed by the routing queue from  
77765 which it is routed. A batch server that has been asked to accept the job may reject the request if  
77766 the job requires resources that are unavailable to that batch server, or if the client is not  
77767 authorized to access the batch server.

77768 Batch servers may route high-priority jobs before low-priority jobs, but, on other than  
77769 overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a  
77770 routing queue reject requests to accept the job for reasons that are permanent, the batch server  
77771 that manages the job shall abort the job. If all or some rejections are temporary, the batch server  
77772 should try to route the job again at some later point.

77773 The reasons for rejecting a batch job are implementation-defined.

77774 The reasons for which the routing should be retried later and the reasons for which the job  
77775 should be aborted are also implementation-defined.

### 77776 3.1.6 Batch Job Execution

77777 To execute a batch job is to create a session leader (a process) that runs the shell program  
77778 indicated by the *Shell\_Path* attribute of the job. The script shall be passed to the program as its  
77779 standard input. An implementation may pass the script to the program by other  
77780 implementation-defined means. At the time a batch job begins execution, it is defined to enter  
77781 the RUNNING state. The primary program that is executed by a batch job is typically, though  
77782 not necessarily, a shell program.

77783 A batch server shall execute eligible jobs as a deferred service—no client request is necessary  
77784 once the batch job is created and eligible. However, the attributes of a batch job, such as the job  
77785 hold type, may render the job ineligible. A batch server shall scan the execution queues that it  
77786 manages for jobs that are eligible for execution. The algorithm by which the batch server selects  
77787 eligible jobs for execution is implementation-defined.

77788 As part of creating the process for the batch job, the batch server shall open the standard output  
77789 and standard error streams of the session.

77790 The attributes of a batch job may indicate that the batch server executing the job shall send mail  
77791 to a list of users at the time it begins execution of the job.

### 77792 3.1.7 Batch Job Exit

77793 When the session leader of an executing job terminates, the job exits. As part of exiting a batch  
77794 job, the batch server that manages the job shall remove the job from the batch queue in which it  
77795 resides. The server shall transfer output files of the job to a location described by the attributes of  
77796 the job.

77797 The attributes of a batch job may indicate that the batch server managing the job shall send mail  
77798 to a list of users at the time the job exits.

### 77799 3.1.8 Batch Job Abort

77800 A batch server shall abort jobs for which a required deferred service cannot be performed. The  
77801 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a  
77802 list of users at the time it aborts the job.

### 77803 3.1.9 Batch Authorization

77804 Clients, such as the batch environment utilities (marked BE), access batch services by means of  
77805 requests to one or more batch servers. To acquire the services of any given batch server, the user  
77806 identifier under which the client runs must be authorized to use that batch server.

77807 The user with an associated user name that creates a batch job shall own the job and can perform  
77808 actions such as read, modify, delete, and move.

77809 A user identifier of the same value at a different host need not be the same user. For example,  
77810 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at  
77811 host **beta**. Likewise, the same person may have access to different user names on different hosts.

77812 An implementation may optionally provide an authorization mechanism that permits one user  
77813 name to access jobs under another user name.

77814 A process on a client host may be authorized to run processes under multiple user names at a  
77815 batch server host. Where appropriate, the utilities defined in this volume of POSIX.1-2017  
77816 provide a means for a user to choose from among such user names when creating or modifying  
77817 a batch job.

### 77818 3.1.10 Batch Administration

77819 The processing of a batch job by a batch server is affected by the attributes of the job. The  
77820 processing of a batch job may also be affected by the attributes of the batch queue in which the  
77821 job resides and by the status of the batch server that manages the job. See also XBD [Chapter 3](#)  
77822 (on page 33) for batch definitions.

77823 **3.1.11 Batch Notification**

77824 Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub*  
 77825 utility creates a batch job and exits. For this reason, batch servers notify users of batch job events  
 77826 by sending mail to the user that owns the job, or to other designated users.

77827 **3.2 Batch Services**

77828 The presence of Batch Environment Services and Utilities option services is indicated by the  
 77829 configuration variable `POSIX2_PBS`. A conforming batch server provides services as defined in  
 77830 this section.

77831 A batch server shall provide batch services in two ways:

- 77832 1. The batch server provides a service at the request of a client.
- 77833 2. The batch server provides a deferred service as a result of a change in conditions  
 77834 monitored by the batch server.

77835 If a batch server cannot complete a request, it shall reject the request. If a batch server cannot  
 77836 complete a deferred service for a batch job, the batch server shall abort the batch job. [Table 3-2](#) is  
 77837 a summary of environment variables that shall be supported by an implementation of the batch  
 77838 server and utilities.

77839 **Table 3-2 Environment Variable Summary**

| Variable                     | Description                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------|
| 77840 <i>PBS_DPREFIX</i>     | 77841 Defines the directive prefix (see <i>qsub</i> )                                   |
| 77842 <i>PBS_ENVIRONMENT</i> | 77843 Batch Job is batch or interactive (see <a href="#">Section 3.2.2.1</a> )          |
| 77844 <i>PBS_JOBID</i>       | 77845 The <i>job_identifier</i> attribute of job (see <a href="#">Section 3.2.3.8</a> ) |
| 77846 <i>PBS_JOBNAME</i>     | 77847 The <i>job_name</i> attribute of job (see <a href="#">Section 3.2.3.8</a> )       |
| 77848 <i>PBS_O_HOME</i>      | 77849 Defines the <i>HOME</i> of the batch client (see <i>qsub</i> )                    |
| 77850 <i>PBS_O_HOST</i>      | 77851 Defines the host name of the batch client (see <i>qsub</i> )                      |
| 77852 <i>PBS_O_LANG</i>      | 77853 Defines the <i>LANG</i> of the batch client (see <i>qsub</i> )                    |
| 77854 <i>PBS_O_LOGNAME</i>   | 77855 Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i> )                 |
| 77856 <i>PBS_O_MAIL</i>      | 77857 Defines the <i>MAIL</i> of the batch client (see <i>qsub</i> )                    |
| 77858 <i>PBS_O_PATH</i>      | 77859 Defines the <i>PATH</i> of the batch client (see <i>qsub</i> )                    |
| 77860 <i>PBS_O_QUEUE</i>     | 77861 Defines the submit queue of the batch client (see <i>qsub</i> )                   |
| 77862 <i>PBS_O_SHELL</i>     | 77863 Defines the <i>SHELL</i> of the batch client (see <i>qsub</i> )                   |
| 77864 <i>PBS_O_TZ</i>        | 77865 Defines the <i>TZ</i> of the batch client (see <i>qsub</i> )                      |
| 77866 <i>PBS_O_WORKDIR</i>   | 77867 Defines the working directory of the batch client (see <i>qsub</i> )              |
| 77868 <i>PBS_QUEUE</i>       | 77869 Defines the initial execution queue (see <a href="#">Section 3.2.2.1</a> )        |

77856 **3.2.1 Batch Job States**

77857 A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD,  
77858 WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests  
77859 that the batch server that manages the batch job can accept for the batch job. A batch server shall  
77860 change the state of a batch job either in response to service requests from clients or as a result of  
77861 deferred services, such as job execution or job routing.

77862 A batch job that is in the QUEUED state resides in a queue but is still pending either execution  
77863 or routing, depending on the queue type.

77864 A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED  
77865 state. A batch server that puts a batch job in an execution queue, but has not yet executed the  
77866 batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution  
77867 queue and is executing is defined to be in the RUNNING state. While a batch job is in the  
77868 RUNNING state, a session leader is associated with the batch job.

77869 A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute,  
77870 is defined to be in the HELD state.

77871 A batch job that is not held, but must wait until a future date and time before executing, is  
77872 defined to be in the WAITING state.

77873 When the session leader associated with a running job exits, the batch job shall be placed in the  
77874 EXITING state.

77875 A batch job for which the session leader has terminated is defined to be in the EXITING state,  
77876 and the batch server that manages such a batch job cannot accept job modification requests that  
77877 affect the batch job. While a batch job is in the EXITING state, the batch server that manages the  
77878 batch job is staging output files and notifying clients of job completion. Once a batch job has  
77879 exited, it no longer exists as an object managed by a batch server.

77880 A batch job that is being moved from a routing queue to another queue is defined to be in the  
77881 TRANSITING state.

77882 When a batch job in a routing queue has been selected to be moved to a new destination, then  
77883 the batch job shall be in either the QUEUED state or the TRANSITING state, depending on the  
77884 batch server implementation.

77885 Batch jobs with either an *Execution\_Time* attribute value set in the future or a *Hold\_Types* attribute  
77886 of value not equal to NO\_HOLD, or both, may be routed or held in the routing queue. The  
77887 treatment of jobs with the *Execution\_Time* or *Hold\_Types* attributes in a routing queue is  
77888 implementation-defined.

77889 When a batch job in a routing queue has not been selected to be moved to a new destination and  
77890 the batch job has a *Hold\_Types* attribute value of other than NO\_HOLD, then the job should be in  
77891 the HELD state.

77892 **Note:** The effect of a hold upon a batch job in a routing queue is implementation-defined. The  
77893 implementation should use the state that matches whether the batch job can route with a hold  
77894 or not.

77895 When a batch job in a routing queue has not been selected to be moved to a new destination and  
77896 the batch job has:

77897 *A Hold\_Types* attribute value of NO\_HOLD

77898 *An Execution\_Time* attribute in the past

77899 then the batch job shall be in the QUEUED state.

77900 When a batch job in a routing queue has not been selected to be moved to a new destination and  
 77901 the batch job has:

77902 A *Hold\_Types* attribute value of NO\_HOLD

77903 An *Execution\_Time* attribute in the future

77904 then the batch job may be in the WAITING state.

77905 **Note:** The effect of a future execution time upon a batch job in a routing queue is implementation-  
 77906 defined. The implementation should use the state that matches whether the batch job can route  
 77907 with a hold or not.

77908 [Table 3-3](#) describes the next state of a batch job, given the current state of the batch job and the  
 77909 type of request. [Table 3-4](#) (on page 2435) describes the response of a batch server to a request,  
 77910 given the current state of the batch job and the type of request.

77911 **3.2.2 Deferred Batch Services**

77912 This section describes the deferred services performed by batch servers: job execution, job  
 77913 routing, job exit, job abort, and the rerunning of jobs after a restart.

77914 **3.2.2.1 Batch Job Execution**

77915 To execute a batch job is to create a session leader (a process) that runs the shell program  
 77916 indicated by the *Shell\_Path\_List* attribute of the batch job. The script is passed to the program as  
 77917 its standard input. An implementation may pass the script to the program by other  
 77918 implementation-defined means. At the time a batch job begins execution, it is defined to enter  
 77919 the RUNNING state.

77920 **Table 3-3** Next State Table

| Request Type               | Current State |   |     |       |   |   |   |
|----------------------------|---------------|---|-----|-------|---|---|---|
|                            | X             | Q | R   | H     | W | E | T |
| Queue Batch Job Request    | Q             | e | e   | e     | e | e | e |
| Modify Batch Job Request   | e             | Q | R   | H     | W | e | T |
| Delete Batch Job Request   | e             | X | E   | X     | X | E | X |
| Batch Job Message Request  | e             | Q | R   | H     | W | E | T |
| Rerun Batch Job Request    | e             | e | Q   | e     | e | e | e |
| Signal Batch Job Request   | e             | e | R   | H     | W | e | e |
| Batch Job Status Request   | e             | Q | R   | H     | W | E | T |
| Batch Queue Status Request | X             | Q | R   | H     | W | E | T |
| Server Status Request      | X             | Q | R   | H     | W | E | T |
| Select Batch Jobs Request  | X             | Q | R   | H     | W | E | T |
| Move Batch Job Request     | e             | Q | R   | H     | W | e | T |
| Hold Batch Job Request     | e             | H | R/H | H     | H | e | T |
| Release Batch Job Request  | e             | Q | R   | Q/W/H | W | e | T |
| Server Shutdown Request    | X             | Q | Q   | H     | W | E | T |
| Locate Batch Job Request   | e             | Q | R   | H     | W | E | T |



77938 **Legend**

77939 X Nonexistent

77940 Q QUEUED

77941 R RUNNING

77942 H HELD

77943 W WAITING

77944 E EXITING

77945 T TRANSITING

77946 e Error

77947 A batch server that has an execution queue containing jobs is said to own the queue and manage  
77948 the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in  
77949 the execution queues owned by the batch server. The batch server shall schedule for execution  
77950 those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling  
77951 jobs is implementation-defined.

77952 A batch server that executes a batch job shall create, in the environment of the session leader of  
77953 the batch job, an environment variable named *PBS\_ENVIRONMENT*, the value of which is the  
77954 string *PBS\_BATCH* encoded in the portable character set.

77955 A batch server that executes a batch job shall create, in the environment of the session leader of  
77956 the batch job, an environment variable named *PBS\_QUEUE*, the value of which is the name of  
77957 the execution queue of the batch job encoded in the portable character set.

77958 To rerun a batch job is to requeue a batch job that is currently executing and then kill the session  
77959 leader of the executing job by sending a SIGKILL prior to completion; see [Section 3.2.3.11](#) (on  
77960 page 2447). A batch server that reruns a batch job shall append the standard output and  
77961 standard error files of the batch job to the corresponding files of the previous execution, if they  
77962 exist, with appropriate annotation. If either file does not exist, that file shall be created as in  
77963 normal execution.

77964

**Table 3-4** Results/Output Table

77965

77966

77967

77968

77969

77970

77971

77972

77973

77974

77975

77976

77977

77978

77979

77980

77981

| Request Type               | Current State |   |   |   |   |   |   |
|----------------------------|---------------|---|---|---|---|---|---|
|                            | X             | Q | R | H | W | E | T |
| Queue Batch Job Request    | O             | e | e | e | e | e | e |
| Modify Batch Job Request   | e             | O | e | O | O | e | e |
| Delete Batch Job Request   | e             | O | O | O | O | e | O |
| Batch Job Message Request  | e             | e | O | e | e | e | e |
| Rerun Batch Job Request    | e             | e | O | e | e | e | e |
| Signal Batch Job Request   | e             | e | O | e | e | e | e |
| Batch Job Status Request   | e             | O | O | O | O | O | O |
| Batch Queue Status Request | O             | O | O | O | O | O | O |
| Server Status Request      | O             | O | O | O | O | O | O |
| Select Batch Job Request   | e             | O | O | O | O | O | O |
| Move Batch Job Request     | e             | O | O | O | O | e | e |
| Hold Batch Job Request     | e             | O | O | O | O | e | e |
| Release Batch Job Request  | e             | O | e | O | O | e | e |
| Server Shutdown Request    | O             | O | e | O | O | e | e |
| Locate Batch Job Request   | e             | O | O | O | O | O | O |

77982

**Legend**

77983

O OK

77984

e Error message

77985

The execution of a batch job by a batch server shall be controlled by job, queue, and server attributes, as defined in this section.

77986

77987

**Account\_Name Attribute**

77988

Batch accounting is an optional feature of batch servers. If a batch server implements accounting, the statements in this section apply and the configuration variable `POSIX2_PBS_ACCOUNTING` shall be set to 1.

77989

77990

77991

A batch server that executes a batch job shall charge the account named in the *Account\_Name* attribute of the batch job for resources consumed by the batch job.

77992

77993

If the *Account\_Name* attribute of the batch job is absent from the batch job attribute list or is altered while the batch job is in execution, the batch server action is implementation-defined.

77994

77995

**Checkpoint Attribute**

77996

Batch checkpointing is an optional feature of batch servers. If a batch server implements checkpointing, the statements in this section apply and the configuration variable `POSIX2_PBS_CHECKPOINT` shall be set to 1.

77997

77998

77999

There are two attributes associated with the checkpointing feature: *Checkpoint* and *Minimum\_Cpu\_Interval*. *Checkpoint* is a batch job attribute, while *Minimum\_Cpu\_Interval* is a queue attribute. An implementation that does not support checkpointing shall support the *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute to other servers.

78000

78001

78002

78003

78004

The behavior of a batch server that executes a batch job for which the value of the *Checkpoint* attribute is `CHECKPOINT_UNSPECIFIED` is implementation-defined. A batch server that executes a batch job for which the value of the *Checkpoint* attribute is `NO_CHECKPOINT` shall

78005

78006

78007 not checkpoint the batch job.

78008 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
78009 CHECKPOINT\_AT\_SHUTDOWN shall checkpoint the batch job only when the batch server  
78010 accepts a request to shut down during the time when the batch job is in the RUNNING state.

78011 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is  
78012 CHECKPOINT\_AT\_MIN\_CPU\_INTERVAL shall checkpoint the batch job at the interval  
78013 specified by the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job has been  
78014 selected. The *Minimum\_Cpu\_Interval* attribute shall be specified in units of CPU minutes.

78015 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an  
78016 unsigned integer shall checkpoint the batch job at an interval that is the value of either the  
78017 *Checkpoint* attribute, or the *Minimum\_Cpu\_Interval* attribute of the queue for which the batch job  
78018 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When  
78019 the *Minimum\_Cpu\_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall  
78020 write a warning message to the standard error stream of the batch job.

### 78021 **Error\_Path Attribute**

78022 The *Error\_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When  
78023 the *Join\_Path* attribute of the batch job is set to the value FALSE and the *Keep\_Files* attribute of  
78024 the batch job does not contain the value KEEP\_STD\_ERROR, a batch server that executes a batch  
78025 job shall perform one of the following actions:

78026         Set the standard error stream of the session leader of the batch job to the path described by  
78027         the value of the *Error\_Path* attribute of the batch job.

78028         Buffer the standard error of the session leader of the batch job until completion of the batch  
78029         job, and when the batch job exits return the contents to the destination described by the  
78030         value of the *Error\_Path* attribute of the batch job.

78031 Applications shall not rely on having access to the standard error of a batch job prior to the  
78032 completion of the batch job.

78033 When the *Error\_Path* attribute does not specify a host name, then the batch server shall retain the  
78034 standard error of the batch job on the host of execution.

78035 When the *Error\_Path* attribute does specify a host name and the *Keep\_Files* attribute does not  
78036 contain the value KEEP\_STD\_ERROR, then the final destination of the standard error of the  
78037 batch job shall be on the host whose host name is specified.

78038 If the path indicated by the value of the *Error\_Path* attribute of the batch job is a relative path, the  
78039 batch server shall expand the path relative to the home directory of the user on the host to which  
78040 the file is being returned.

78041 When the batch server buffers the standard error of the batch job and the file cannot be opened  
78042 for write upon completion of the batch job, then the server shall place the standard error in an  
78043 implementation-defined location and notify the user of the location via mail. It shall be possible  
78044 for the user to process this mail using the *mailx* utility.

78045 If a batch server that does not buffer the standard error cannot open the standard error path of  
78046 the batch job for write access, then the batch server shall abort the batch job.

78047 **Execution\_Time Attribute**

78048 A batch server shall not execute a batch job before the time represented by the value of the  
78049 *Execution\_Time* attribute of the batch job. The *Execution\_Time* attribute is defined in seconds since  
78050 the Epoch.

78051 **Hold\_Types Attribute**

78052 A batch server shall support the following hold types:

- 78053 s Can be set or released by a user with at least a privilege level of batch administrator  
78054 (SYSTEM).
- 78055 o Can be set or released by a user with at least a privilege level of batch operator  
78056 (OPERATOR).
- 78057 u Can be set or released by the user with at least a privilege level of user, where the user is  
78058 defined in the *Job\_Owner* attribute (USER).
- 78059 n Indicates that none of the *Hold\_Types* attributes are set (NO\_HOLD).

78060 An implementation may define other hold types. Any additional hold types, how they are  
78061 specified, their internal representation, their behavior, and how they affect the behavior of other  
78062 utilities are implementation-defined.

78063 The value of the *Hold\_Types* attribute shall be the union of the valid hold types ('s', 'o', 'u',  
78064 and any implementation-defined hold types), or 'n'.

78065 A batch server shall not execute a batch job if the *Hold\_Types* attribute of the batch job has a  
78066 value other than NO\_HOLD. If the *Hold\_Types* attribute of the batch job has a value other than  
78067 NO\_HOLD, the batch job shall be in the HELD state.

78068 **Job\_Owner Attribute**

78069 The *Job\_Owner* attribute consists of a pair of user name and host name values of the form:

78070 `username@hostname`

78071 A batch server that accepts a *Queue Batch Job Request* shall set the *Job\_Owner* attribute to a string  
78072 that is the `username@hostname` of the user who submitted the job.

78073 **Join\_Path Attribute**

78074 A batch server that executes a batch job for which the value of the *Join\_Path* attribute is TRUE  
78075 shall ignore the value of the *Error\_Path* attribute and merge the standard error of the batch job  
78076 with the standard output of the batch job.

78077 **Keep\_Files Attribute**

78078 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
78079 the value KEEP\_STD\_OUTPUT shall retain the standard output of the batch job on the host  
78080 where execution occurs. The standard output shall be retained in the home directory of the user  
78081 under whose user ID the batch job is executed and the filename shall be the default filename for  
78082 the standard output as defined under the `-o` option of the *qsub* utility. The *Output\_Path* attribute  
78083 is not modified.

78084 A batch server that executes a batch job for which the value of the *Keep\_Files* attribute includes  
78085 the value KEEP\_STD\_ERROR shall retain the standard error of the batch job on the host where  
78086 execution occurs. The standard error shall be retained in the home directory of the user under  
78087 whose user ID the batch job is executed and the filename shall be the default filename for

78088 standard error as defined under the `-e` option of the `qsub` utility. The `Error_Path` attribute is not  
78089 modified.

78090 A batch server that executes a batch job for which the value of the `Keep_Files` attribute includes  
78091 values other than `KEEP_STD_OUTPUT` and `KEEP_STD_ERROR` shall retain these other files on  
78092 the host where execution occurs. These files (with implementation-defined names) shall be  
78093 retained in the home directory of the user under whose user identifier the batch job is executed.

#### 78094 **Mail\_Points and Mail\_Users Attributes**

78095 A batch server that executes a batch job for which one of the values of the `Mail_Points` attribute is  
78096 the value `MAIL_AT_BEGINNING` shall send a mail message to each user account listed in the  
78097 `Mail_Users` attribute of the batch job.

78098 The mail message shall contain at least the batch job identifier, queue, and server at which the  
78099 batch job currently resides, and the `Job_Owner` attribute.

#### 78100 **Output\_Path Attribute**

78101 The `Output_Path` attribute of a running job cannot be changed by a *Modify Batch Job Request*.  
78102 When the `Keep_Files` attribute of the batch job does not contain the value `KEEP_STD_OUTPUT`, a  
78103 batch server that executes a batch job shall either:

78104       Set the standard output stream of the session leader of the batch job to the destination  
78105       described by the value of the `Output_Path` attribute of the batch job.

78106       or:

78107       Buffer the standard output of the session leader of the batch job until completion of the  
78108       batch job, and when the batch job exits return the contents to the destination described by  
78109       the value of the `Output_Path` attribute of the batch job.

78110 When the `Output_Path` attribute does not specify a host name, then the batch server shall retain  
78111 the standard output of the batch job on the host of execution.

78112 When the `Keep_Files` attribute does not contain the value `KEEP_STD_OUTPUT` and the  
78113 `Output_Path` attribute does specify a host name, then the final destination of the standard output  
78114 of the batch job shall be on the host specified.

78115 If the path specified in the `Output_Path` attribute of the batch job is a relative path, the batch  
78116 server shall expand the path relative to the home directory of the user on the host to which the  
78117 file is being returned.

78118 Whether or not the batch server buffers the standard output of the batch job until completion of  
78119 the batch job is implementation-defined. Applications shall not rely on having access to the  
78120 standard output of a batch job prior to the completion of the batch job.

78121 When the batch server does buffer the standard output of the batch job and the file cannot be  
78122 opened for write upon completion of the batch job, then the batch server shall place the standard  
78123 output in an implementation-defined location and notify the user of the location via mail. It shall  
78124 be possible for the user to process this mail using the `mailx` utility.

78125 If a batch server that does not buffer the standard output cannot open the standard output path  
78126 of the batch job for write access, then the batch server shall abort the batch job.

**78127 Priority Attribute**

78128 A batch server implementation may choose to preferentially execute a batch job based on the  
78129 *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is  
78130 implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger  
78131 values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

**78132 Rerunable Attribute**

78133 A batch job that began execution but did not complete, because the batch server either shut  
78134 down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has  
78135 the value TRUE.

78136 If a batch job, which was requeued after beginning execution but prior to completion, has a valid  
78137 checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted  
78138 from the last valid checkpoint.

78139 If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable*  
78140 attribute value of TRUE and was requeued after beginning execution but prior to completion,  
78141 the batch server shall place the batch job into execution at the beginning of the job.

78142 When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after  
78143 beginning execution but prior to completion, and the batch job cannot be restarted from a  
78144 checkpoint, then the batch server shall abort the batch job.

**78145 Resource\_List Attribute**

78146 A batch server that executes a batch job shall establish the resource limits of the session leader of  
78147 the batch job according to the values of the *Resource\_List* attribute of the batch job. Resource  
78148 limits shall be enforced by an implementation-defined method.

**78149 Shell\_Path\_List Attribute**

78150 The *Shell\_Path\_List* job attribute consists of a list of pairs of pathname and host name values. The  
78151 host name component can be omitted, in which case the pathname serves as the default  
78152 pathname when a batch server cannot find the name of the host on which it is running in the list.

78153 A batch server that executes a batch job shall select, from the value of the *Shell\_Path\_List*  
78154 attribute of the batch job, a pathname where the shell to execute the batch job shall be found.  
78155 The batch server shall select the pathname, in order of preference, according to the following  
78156 methods:

78157         Select the pathname that contains the name of the host on which the batch server is  
78158         running.

78159         Select the pathname for which the host name has been omitted.

78160         Select the pathname for the login shell of the user under which the batch job is to execute.

78161 If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

78162 If the value of the selected pathname from the *Shell\_Path\_List* attribute of the batch job  
78163 represents a partial path, the batch server shall expand the path relative to a path that is  
78164 implementation-defined.

78165 The batch server that executes the batch job shall execute the program that was selected from the  
78166 *Shell\_Path\_List* attribute of the batch job. The batch server shall pass the path to the script of the  
78167 batch job as the first argument to the shell program.

78168 **User\_List Attribute**

78169 The *User\_List* job attribute consists of a list of pairs of user name and host name values. The host  
 78170 name component can be omitted, in which case the user name serves as a default when a batch  
 78171 server cannot find the name of the host on which it is running in the list.

78172 A batch server that executes a batch job shall select, from the value of the *User\_List* attribute of  
 78173 the batch job, a user name under which to create the session leader. The server shall select the  
 78174 user name, in order of preference, according to the following methods:

78175         Select the user name of a value that contains the name of the host on which the batch  
 78176 server executes.

78177         Select the user name of a value for which the host name has been omitted.

78178         Select the user name from the *Job\_Owner* attribute of the batch job.

78179 **Variable\_List Attribute**

78180 A batch server that executes a batch job shall create, in the environment of the session leader of  
 78181 the batch job, each environment variable listed in the *Variable\_List* attribute of the batch job, and  
 78182 set the value of each such environment variable to that of the corresponding variable in the  
 78183 variable list.

78184 3.2.2.2 *Batch Job Routing*

78185 To route a batch job is to select a queue from a list and move the batch job to that queue.

78186 A batch server that has routing queues, which have been started, shall route the jobs in the  
 78187 routing queues owned by the batch server. A batch server may delay the routing of a batch job.  
 78188 The algorithm for selecting a batch job and the queue to which it will be routed is  
 78189 implementation-defined.

78190 When a routing queue has multiple possible destinations specified, then the precedence of the  
 78191 destinations is implementation-defined.

78192 A batch server that routes a batch job to a queue at another server shall move the batch job into  
 78193 the target queue with a *Queue Batch Job Request*.

78194 If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the  
 78195 batch job or abort the batch job. A batch server that retries failed routings shall provide a means  
 78196 for the batch administrator to specify the number of retries and the minimum period of time  
 78197 between retries. The means by which an administrator specifies the number of retries and the  
 78198 delay between retries is implementation-defined. When the number of retries specified by the  
 78199 batch administrator has been exhausted, the batch server shall abort the batch job and perform  
 78200 the functions of *Batch Job Exit*; see [Section 3.2.2.3](#).

78201 3.2.2.3 *Batch Job Exit*

78202 For each job in the EXITING state, the batch server that exited the batch job shall perform the  
 78203 following deferred services in the order specified:

78204         1. If buffering standard error, move that file into the location specified by the *Error\_Path*  
 78205 attribute of the batch job.

78206         2. If buffering standard output, move that file into the location specified by the *Output\_Path*  
 78207 attribute of the batch job.

78208 3. If the *Mail\_Points* attribute of the batch job includes MAIL\_AT\_EXIT, send mail to the  
 78209 users listed in the *Mail\_Users* attribute of the batch job. The mail message shall contain at  
 78210 least the batch job identifier, queue, and server at which the batch job currently resides,  
 78211 and the *Job\_Owner* attribute.

78212 4. Remove the batch job from the queue.

78213 If a batch server that buffers the standard error output cannot return the standard error file to  
 78214 the standard error path at the time the batch job exits, the batch server shall do one of the  
 78215 following:

78216 Mail the standard error file to the batch job owner.

78217 Save the standard error file and mail the location and name of the file where the standard  
 78218 error is stored to the batch job owner.

78219 Save the standard error file and notify the user by other implementation-defined means.

78220 If a batch server that buffers the standard output cannot return the standard output file to the  
 78221 standard output path at the time the batch job exits, the batch server shall do one of the  
 78222 following:

78223 Mail the standard output file to the batch job owner.

78224 Save the standard output file and mail the location and name of the file where the standard  
 78225 output is stored to the batch job owner.

78226 Save the standard output file and notify the user by other implementation-defined means.

78227 At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

#### 78228 3.2.2.4 *Batch Server Restart*

78229 A batch server that has been either shutdown or terminated abnormally, and has returned to  
 78230 operation, is said to have “restarted”.

78231 Upon restarting, a batch server shall requeue those jobs managed by the batch server that were  
 78232 in the RUNNING state at the time the batch server shut down and for which the *Rerunable*  
 78233 attribute of the batch job has the value TRUE.

78234 Queues are defined to be non-volatile. A batch server shall store the content of queues that it  
 78235 controls in such a way that server and system shutdowns do not erase the content of the queues.

#### 78236 3.2.2.5 *Batch Job Abort*

78237 A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

78238 A batch server that aborts a batch job shall perform the following services:

78239 Delete the batch job from the queue in which it resides.

78240 If the *Mail\_Points* attribute of the batch job includes the value MAIL\_AT\_ABORT, send  
 78241 mail to the users listed in the value of the *Mail\_Users* attribute of the job. The mail message  
 78242 shall contain at least the batch job identifier, queue, and server at which the batch job  
 78243 currently resides, the *Job\_Owner* attribute, and the reason for the abort.

78244 If the batch job was in the RUNNING state, terminate the session leader of the executing  
 78245 job by sending the session leader a SIGKILL, place the batch job in the EXITING state, and  
 78246 perform the actions of *Batch Job Exit*.



78247 **3.2.3 Requested Batch Services**

78248 This section describes the services provided by batch servers in response to requests from  
 78249 clients. [Table 3-5](#) summarizes the current set of batch service requests and for each gives its type  
 78250 (deferred or not) and whether it is an optional function.

78251 **Table 3-5** Batch Services Summary

| Batch Service                     | Deferred | Optional |
|-----------------------------------|----------|----------|
| <i>Batch Job Execution</i>        | Yes      | No       |
| <i>Batch Job Routing</i>          | Yes      | No       |
| <i>Batch Job Exit</i>             | Yes      | No       |
| <i>Batch Server Restart</i>       | Yes      | No       |
| <i>Batch Job Abort</i>            | Yes      | No       |
| <i>Delete Batch Job Request</i>   | No       | No       |
| <i>Hold Batch Job Request</i>     | No       | No       |
| <i>Batch Job Message Request</i>  | No       | Yes      |
| <i>Batch Job Status Request</i>   | No       | No       |
| <i>Locate Batch Job Request</i>   | No       | Yes      |
| <i>Modify Batch Job Request</i>   | No       | No       |
| <i>Move Batch Job Request</i>     | No       | No       |
| <i>Queue Batch Job Request</i>    | No       | No       |
| <i>Batch Queue Status Request</i> | No       | No       |
| <i>Release Batch Job Request</i>  | No       | No       |
| <i>Rerun Batch Job Request</i>    | No       | No       |
| <i>Select Batch Jobs Request</i>  | No       | No       |
| <i>Server Shutdown Request</i>    | No       | No       |
| <i>Server Status Request</i>      | No       | No       |
| <i>Signal Batch Job Request</i>   | No       | No       |
| <i>Track Batch Job Request</i>    | No       | Yes      |

78274 If a request is rejected because the batch client is not authorized to perform the action, the batch  
 78275 server shall return the same status as when the batch job does not exist.

78276 **3.2.3.1** *Delete Batch Job Request*

78277 A batch job is defined to have been deleted when it has been removed from the queue in which  
 78278 it resides and not instantiated in another queue. A client requests that the server that manages a  
 78279 batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

78280 A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

78281       The user of the batch client is not authorized to delete the designated job.

78282       The designated job is not managed by the batch server.

78283       The designated job is in a state inconsistent with the delete request.

78284 A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons.  
 78285 The method used to determine whether the user of a client is authorized to perform the  
 78286 requested action is implementation-defined.

78287 A batch server requested to delete a batch job shall delete the batch job if the batch job exists and  
 78288 is not in the EXITING state.

78289 A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the

78290 session leader of the batch job. It is implementation-defined whether additional signals are sent  
78291 to the session leader of the job prior to sending the SIGKILL signal.

78292 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the  
78293 EXITING state after it has killed the session leader of the batch job and shall perform the actions  
78294 of *Batch Job Exit*.

#### 78295 3.2.3.2 *Hold Batch Job Request*

78296 A batch client can request that the batch server add one or more holds to a batch job. Such a  
78297 request is called a *Hold Batch Job Request*.

78298 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

78299         The batch server does not support one or more of the requested holds to be added to the  
78300 batch job.

78301         The user of the batch client is not authorized to add one or more of the requested holds to  
78302 the batch job.

78303         The batch server does not manage the specified job.

78304         The designated job is in the EXITING state.

78305 A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The  
78306 method used to determine whether the user of a client is authorized to perform the requested  
78307 action is implementation-defined.

78308 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall  
78309 place a hold on the batch job. The effects, if any, the hold will have on a batch job in the  
78310 RUNNING state are implementation-defined.

78311 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*  
78312 *Batch Job Request*, that is not already present, to the value of the *Hold\_Types* attribute of the batch  
78313 job.

#### 78314 3.2.3.3 *Batch Job Message Request*

78315 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports  
78316 *Batch Job Message Request*, the statements in this section apply and the configuration variable  
78317 POSIX2\_PBS\_MESSAGE shall be set to 1.

78318 A batch client can request that a batch server write a message into certain output files of a batch  
78319 job. Such a request is called a *Batch Job Message Request*.

78320 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

78321         The batch server does not support sending messages to jobs.

78322         The user of the batch client is not authorized to post a message to the designated job.

78323         The designated job does not exist on the batch server.

78324         The designated job is not in the RUNNING state.

78325 A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons.  
78326 The method used to determine whether the user of a client is authorized to perform the  
78327 requested action is implementation-defined.

78328 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch  
78329 client into the files indicated by the batch client.

78330 3.2.3.4 *Batch Job Status Request*

78331 A batch client can request that a batch server respond with the status and attributes of a batch  
78332 job. Such a request is called a *Batch Job Status Request*.

78333 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

78334       The user of the batch client is not authorized to query the status of the designated job.

78335       The designated job is not managed by the batch server.

78336 A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons.  
78337 The method used to determine whether the user of a client is authorized to perform the  
78338 requested action is implementation-defined.

78339 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the  
78340 batch client.

78341 A batch server may return other information in response to a *Batch Job Status Request*.

78342 3.2.3.5 *Locate Batch Job Request*

78343 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports  
78344 *Locate Batch Job Request*, the statements in this section apply and the configuration variable  
78345 POSIX2\_PBS\_LOCATE shall be set to 1.

78346 A batch client can ask a batch server to respond with the location of a batch job that was created  
78347 by the batch server. Such a request is called a *Locate Batch Job Request*.

78348 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to  
78349 the batch client.

78350 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that  
78351 server.

78352 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by  
78353 that server; that is, for a batch job that is not in a queue owned by that server.

78354 A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons.

78355 3.2.3.6 *Modify Batch Job Request*

78356 Batch clients modify (alter) the attributes of a batch job by making a request to the server that  
78357 manages the batch job. Such a request is called a *Modify Batch Job Request*.

78358 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

78359       The user of the batch client is not authorized to make the requested modification to the  
78360 batch job.

78361       The designated job is not managed by the batch server.

78362       The requested modification is inconsistent with the state of the batch job.

78363       An unrecognized resource is requested for a batch job in an execution queue.

78364 A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons.  
78365 The method used to determine whether the user of a client is authorized to perform the  
78366 requested action is implementation-defined.

78367 A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of  
78368 the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the

- 78369 attributes of the batch job.
- 78370 If the servicing by a batch server of an otherwise valid request would result in no change, then  
78371 the batch server shall indicate successful completion of the request.
- 78372 3.2.3.7 *Move Batch Job Request*
- 78373 A batch client can request that a batch server move a batch job to another destination. Such a  
78374 request is called a *Move Batch Job Request*.
- 78375 A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:
- 78376 The user of the batch client is not authorized to remove the designated job from the queue  
78377 in which the batch job resides.
- 78378 The user of the batch client is not authorized to move the designated job to the destination.
- 78379 The designated job is not managed by the batch server.
- 78380 The designated job is in the EXITING state.
- 78381 The destination is inaccessible.
- 78382 A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The  
78383 method used to determine whether the user of a client is authorized to perform the requested  
78384 action is implementation-defined.
- 78385 A batch server that accepts a *Move Batch Job Request* shall perform the following services:
- 78386 Queue the designated job at the destination.
- 78387 Remove the designated job from the queue in which the batch job resides.
- 78388 If the destination resides on another batch server, the batch server shall queue the batch job at  
78389 the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*  
78390 *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job*  
78391 *Request* succeeds, the batch server shall remove the batch job from its queue.
- 78392 The batch server shall not modify any attributes of the batch job.
- 78393 3.2.3.8 *Queue Batch Job Request*
- 78394 A batch queue is controlled by one and only one batch server. A batch server is said to own the  
78395 queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a  
78396 request is called a *Queue Batch Job Request*.
- 78397 A batch server requested to queue a batch job for which the queue is not specified shall select an  
78398 implementation-defined queue for the batch job. Such a queue is called the “default queue” of  
78399 the batch server. The implementation shall provide the means for a batch administrator to  
78400 specify the default queue. The queue, whether specified or defaulted, is called the “target  
78401 queue”.
- 78402 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:
- 78403 The client is not authorized to create a batch job in the target queue.
- 78404 The request specifies a queue that does not exist on the batch server.
- 78405 The target queue is an execution queue and the batch server cannot satisfy a resource  
78406 requirement of the batch job.

- 78407 The target queue is an execution queue and an unrecognized resource is requested.
- 78408 The target queue is an execution queue, the batch server does not support checkpointing,  
78409 and the value of the *Checkpoint* attribute of the batch job is not NO\_CHECKPOINT.
- 78410 The job requires access to a user identifier that the batch client is not authorized to access.
- 78411 A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons.
- 78412 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
78413 PBS\_O\_QUEUE value is missing from the value of the *Variable\_List* attribute of the batch job  
78414 shall add that variable to the list and set the value to the name of the target queue. Once set, no  
78415 server shall change the value of PBS\_O\_QUEUE, even if the batch job is moved to another  
78416 queue.
- 78417 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS\_JOBID  
78418 value is missing from the value of the *Variable\_List* attribute shall add that variable to the list and  
78419 set the value to the batch job identifier assigned by the server in the format:
- 78420 `sequence_number.server`
- 78421 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the  
78422 PBS\_JOBNAME value is missing from the value of the *Variable\_List* attribute of the batch job  
78423 shall add that variable to the list and set the value to the *Job\_Name* attribute of the batch job.
- 78424 3.2.3.9 *Batch Queue Status Request*
- 78425 A batch client can request that a batch server respond with the status and attributes of a queue.  
78426 Such a request is called a *Batch Queue Status Request*.
- 78427 A batch server shall reject a *Batch Queue Status Request* if any of the following statements are  
78428 true:
- 78429 The user of the batch client is not authorized to query the status of the designated queue.
- 78430 The designated queue does not exist on the batch server.
- 78431 A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons.  
78432 The method used to determine whether the user of a client is authorized to perform the  
78433 requested action is implementation-defined.
- 78434 A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to  
78435 the batch client.
- 78436 3.2.3.10 *Release Batch Job Request*
- 78437 A batch client can request that the server remove one or more holds from a batch job. Such a  
78438 request is called a *Release Batch Job Request*.
- 78439 A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:
- 78440 The user of the batch client is not authorized to remove one or more of the requested holds  
78441 from the batch job.
- 78442 The batch server does not manage the specified job.
- 78443 A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons.  
78444 The method used to determine whether the user of a client is authorized to perform the  
78445 requested action is implementation-defined.
- 78446 A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the

78447            *Release Batch Job Request*, that is present, from the value of the *Hold\_Types* attribute of the batch  
78448 job.

78449    3.2.3.11    *Rerun Batch Job Request*

78450            To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible  
78451 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request  
78452 is called *Rerun Batch Job Request*.

78453            A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

78454                    The user of the batch client is not authorized to rerun the designated job.

78455                    The *Rerunable* attribute of the designated job has the value FALSE.

78456                    The designated job is not in the RUNNING state.

78457                    The batch server does not manage the designated job.

78458            A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons.  
78459 The method used to determine whether the user of a client is authorized to perform the  
78460 requested action is implementation-defined.

78461            A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the  
78462 batch job.

78463            A batch server that accepts a request to rerun a batch job shall perform the following services:

78464                    Requeue the batch job in the execution queue in which it was executing.

78465                    Send a SIGKILL signal to the process group of the session leader of the batch job.

78466            An implementation may indicate to the batch job owner that the batch job has been rerun.  
78467 Whether and how the batch job owner is notified that a batch job is rerun is implementation-  
78468 defined.

78469            A batch server that reruns a batch job may send other implementation-defined signals to the  
78470 session leader of the batch job prior to sending the SIGKILL signal.

78471            A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be  
78472 selected for execution before other jobs is implementation-defined.

78473    3.2.3.12    *Select Batch Jobs Request*

78474            A batch client can request from a batch server a list of jobs managed by that server that match a  
78475 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs  
78476 managed by the batch server that receives the request are candidates for selection.

78477            A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job  
78478 identifiers that correspond to jobs that meet the selection criteria.

78479            If the batch client is not authorized to query the status of a batch job, the batch server shall not  
78480 select the batch job.

78481 3.2.3.13 *Server Shutdown Request*

78482 A batch server is defined to have shut down when it does not respond to requests from clients  
78483 and does not perform deferred services for jobs. A batch client can request that a batch server  
78484 shut down. Such a request is called a *Server Shutdown Request*.

78485 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut  
78486 down the batch server. The method used to determine whether the user of a client is authorized  
78487 to perform the requested action is implementation-defined.

78488 A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons.  
78489 The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined.

78490 At server shutdown, a batch server shall do, in order of preference, one of the following:

78491 If checkpointing is implemented and the batch job is checkpointable, then checkpoint the  
78492 batch job and requeue it.

78493 If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the  
78494 beginning).

78495 Abort the batch job.

78496 3.2.3.14 *Server Status Request*

78497 A batch client can request that a batch server respond with the status and attributes of the batch  
78498 server. Such a request is called a *Server Status Request*.

78499 A batch server shall reject a *Server Status Request* if the following statement is true:

78500 The user of the batch client is not authorized to query the status of the designated server.

78501 A batch server may reject a *Server Status Request* for other implementation-defined reasons. The  
78502 method used to determine whether the user of a client is authorized to perform the requested  
78503 action is implementation-defined.

78504 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch  
78505 client.

78506 3.2.3.15 *Signal Batch Job Request*

78507 A batch client can request that a batch server signal the session leader of a batch job. Such a  
78508 request is called a *Signal Batch Job Request*.

78509 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

78510 The user of the batch client is not authorized to signal the batch job.

78511 The job is not in the RUNNING state.

78512 The batch server does not manage the designated job.

78513 The requested signal is not supported by the implementation.

78514 A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons.  
78515 The method used to determine whether the user of a client is authorized to perform the  
78516 requested action is implementation-defined.

78517 A batch server that accepts a request to signal a batch job shall send the signal requested by the  
78518 batch client to the process group of the session leader of the batch job.

78519 3.2.3.16 *Track Batch Job Request*

78520 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports  
78521 *Track Batch Job Request*, the statements in this section apply and the configuration variable  
78522 POSIX2\_PBS\_TRACK shall be set to 1.

78523 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients  
78524 may use the tracking information to determine the batch server that should receive a batch  
78525 server request.

78526 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a  
78527 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that  
78528 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that  
78529 created the job.

78530 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also  
78531 be sent to other servers as a backup to the primary server. The method by which backup servers  
78532 are specified is implementation-defined.

78533 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,  
78534 then the batch server shall record the current location of the batch job as contained in the  
78535 request.

78536 **3.3 Common Behavior for Batch Environment Utilities**78537 **3.3.1 Batch Job Identifier**

78538 A utility shall recognize *job\_identifiers* of the format:

78539 `[sequence_number][.server_name][@server]`

78540 where:

78541 *sequence\_number* An integer that, when combined with *server\_name*, provides a batch job  
78542 identifier that is unique within the batch system.

78543 *server\_name* The name of the batch server to which the batch job was originally submitted.

78544 *server* The name of the batch server that is currently managing the batch job.

78545 If the application omits the batch *server\_name* portion of a batch job identifier, a utility shall use  
78546 the name of a default batch server.

78547 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

78548 The batch server indicated by *server\_name*, if present

78549 The name of the default batch server

78550 The name of the batch server that is currently managing the batch job

78551 If only *@server* is specified, then the status of all jobs owned by the user on the requested server  
78552 is listed.

78553 The means by which a utility determines the default batch server is implementation-defined.

78554 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility  
78555 shall send the request to the specified server.



78556 A strictly conforming application shall use the syntax described for the job identifier. Whenever  
78557 a batch job identifier is specified whose syntax is not recognized by an implementation, then a  
78558 message for each error that occurs shall be written to standard error and the utility shall exit  
78559 with an exit status greater than zero.

78560 When a batch job identifier is supplied as an argument to a batch utility and the *server\_name*  
78561 portion of the batch job identifier is omitted, then the utility shall use the name of the default  
78562 batch server.

78563 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
78564 portion of the batch job identifier is omitted, then the utility shall use either:

78565       The name of the default batch server

78566       or:

78567       The name of the batch server that is currently managing the batch job

78568 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*  
78569 portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*  
78570 *Request* to the specified server.

### 78571 3.3.2 Destination

78572 The utility shall recognize a *destination* of the format:

78573 [queue][@server]

78574 where:

78575 *queue*       The name of a valid execution or routing queue at the batch server denoted by  
78576                @*server*, defined as a string of up to 15 alphanumeric characters in the portable  
78577 character set (see XBD Section 6.1, on page 125) where the first character is  
78578 alphabetic.

78579 *server*       The name of a batch server, defined as a string of alphanumeric characters in the  
78580 portable character set.

78581 If the application omits the batch *server* portion of a destination, then the utility shall use either:

78582       The name of the default batch server

78583       or:

78584       The name of the batch server that is currently managing the batch job

78585 The means by which a utility determines the default batch server is implementation-defined.

78586 If the application omits the *queue* portion of a destination, then the utility shall use the name of  
78587 the default queue at the batch server chosen. The means by which a batch server determines its  
78588 default queue is implementation-defined. If a destination is specified in the *queue@server* form,  
78589 then the utility shall use the specified queue at the specified server.

78590 A strictly conforming application shall use the syntax described for a destination. Whenever a  
78591 destination is specified whose syntax is not recognized by an implementation, then a message  
78592 shall be written to standard error and the utility shall exit with an exit status greater than zero.

78593 **3.3.3 Multiple Keyword-Value Pairs**

78594 For each option that can have multiple keyword-value pair arguments, the following rules shall  
 78595 apply. Examples of options that can have list-oriented option-arguments are `-u value@keyword`  
 78596 and `-l keyword=value`.

78597 1. If a batch utility is presented with a list-oriented option-argument for which a keyword  
 78598 has a corresponding value that begins with a single or double-quote, then the utility shall  
 78599 stop interpreting the input stream for delimiters until a second single or double-quote,  
 78600 respectively, is encountered. This feature allows some flexibility for a <comma> (',') or  
 78601 <equals-sign> ('=') to be part of the value string for a particular keyword; for example:

```
78602 keyword1='val1,val2',keywd2="val3,val4"
```

78603 **Note:** This may require the user to escape the quotes as in the following command:

```
78604 foo -xkeyword1=\'val1,val2\',keywd2=\'val3,val4\'
```

78605 2. If a batch server is presented with a list-oriented attribute that has a keyword that was  
 78606 encountered earlier in the list, then the later entry for that keyword shall replace the  
 78607 earlier entry.

78608 3. If a batch server is presented with a list-oriented attribute that has a keyword without any  
 78609 corresponding value of the form `keyword=` or `@keyword` and the same keyword was  
 78610 encountered earlier in the list, then the prior entry for that keyword shall be ignored by  
 78611 the batch server.

78612 4. If a batch utility is expecting a list-oriented option-argument entry of the form  
 78613 `keyword=value`, but is presented with an entry of the form `keyword` without any  
 78614 corresponding `value`, then the entry shall be treated as though a default value of NULL  
 78615 was assigned (that is, `keyword=NULL`) for entry parsing purposes. The utility shall  
 78616 include only the keyword, not the NULL value, in the associated job attribute.

78617 5. If a batch utility is expecting a list-oriented option-argument entry of the form  
 78618 `value@keyword`, but is presented with an entry of the form `value` without any  
 78619 corresponding `keyword`, then the entry shall be treated as though a keyword of NULL was  
 78620 assigned (that is, `value@NULL`) for entry parsing purposes. The utility shall include only  
 78621 the value, not the NULL keyword, in the associated job attribute.

78622 6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the  
 78623 same keyword, interpreting the keywords, in order, with the last value encountered  
 78624 taking precedence over prior instances of the same keyword. This rule allows, but does  
 78625 not require, a batch utility to preprocess the attribute to remove duplicate keywords.


78626 7. If a batch utility is presented with multiple list-oriented option-arguments on the  
 78627 command line or in script directives, or both, for a single option, then the utility shall  
 78628 concatenate, in order, any command line keyword and value pairs to the end of any  
 78629 directive keyword and value pairs separated by a single <comma> to produce a single  
 78630 string that is an equivalent, valid option-argument. The resulting string shall be assigned  
 78631 to the associated attribute of the batch job (after optionally removing duplicate entries as  
 78632 described in item 6).



78633

Chapter 4

78634



# Utilities

78635

This chapter contains the definitions of the utilities, as follows:

78636

Mandatory utilities that are present on every conformant system

78637

Optional utilities that are present only on systems supporting the associated option; see

78638

[Section 1.7.1](#) (on page 7) for information on the options in this volume of POSIX.1-2017

78639 **NAME**78640 admin — create and administer SCCS files (**DEVELOPMENT**)78641 **SYNOPSIS**

```

78642 XSI admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag]
78643 [-m mrlist] [-r rel] [-t[name] [-y[comment]]] newfile
78644
78645 admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
78646 [-t[name]] [-y[comment]] newfile...
78647
78647 admin -h file...
78648
78648 admin -z file...

```

78649 **DESCRIPTION**

78650 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named  
 78651 file does not exist, it shall be created, and its parameters shall be initialized according to the  
 78652 specified options. Parameters not initialized by an option shall be assigned a default value. If a  
 78653 named file does exist, parameters corresponding to specified options shall be changed, and other  
 78654 parameters shall be left as is.

78655 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files  
 78656 shall be given read-only permission mode. Write permission in the parent directory is required  
 78657 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)  
 78658 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode  
 78659 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file  
 78660 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This  
 78661 ensures that changes are made to the SCCS file only if no errors occur.

78662 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent  
 78663 simultaneous updates to the SCCS file; see *get*.

78664 **OPTIONS**

78665 The *admin* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the *-i*, *-t*, and *-y*  
 78666 options have optional option-arguments. These optional option-arguments shall not be  
 78667 presented as separate arguments. The following options are supported:

78668 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created  
 78669 with control information but without any file data.

78670 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.  
 78671 The text constitutes the first delta of the file (see the *-r* option for the delta  
 78672 numbering scheme). If the *-i* option is used, but the *name* option-argument is  
 78673 omitted, the text shall be obtained by reading the standard input. If this option is  
 78674 omitted, the SCCS file shall be created with control information but without any  
 78675 file data. The *-i* option implies the *-n* option.

78676 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that  
 78677 is, the branch and sequence numbers shall be zero or missing. The level number is  
 78678 optional, and defaults to 1.

78679 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be  
 78680 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):

78681 A **-t** option without a *name* option-argument shall cause the removal of  
 78682 descriptive text (if any) currently in the SCCS file.

78683 A **-t** option with a *name* option-argument shall cause the text (if any) in the  
 78684 named file to replace the descriptive text (if any) currently in the SCCS file.

78685 **-f flag** Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file.  
 78686 Several **-f** options may be supplied on a single *admin* command line.  
 78687 Implementations shall recognize the following flags and associated values:

78688 **b** Allow use of the **-b** option on a *get* command to create branch deltas.

78689 **cceil** Specify the highest release (that is, ceiling), a number less than or equal to  
 78690 9999, which may be retrieved by a *get* command for editing. The default  
 78691 value for an unspecified **c** flag shall be 9999.

78692 **ffloor** Specify the lowest release (that is, floor), a number greater than 0 but less  
 78693 than 9999, which may be retrieved by a *get* command for editing. The  
 78694 default value for an unspecified **f** flag shall be 1.

78695 **dSID** Specify the default delta number (SID) to be used by a *get* command.

78696 **istr** Treat the “No ID keywords” message issued by *get* or *delta* as a fatal error.  
 78697 In the absence of this flag, the message is only a warning. The message is  
 78698 issued if no SCCS identification keywords (see *get*) are found in the text  
 78699 retrieved or stored in the SCCS file. If a value is supplied, the application  
 78700 shall ensure that the keywords exactly match the given string; however,  
 78701 the string shall contain a keyword, and no embedded <newline>  
 78702 characters.

78703 **j** Allow concurrent *get* commands for editing on the same SID of an SCCS  
 78704 file. This allows multiple concurrent updates to the same version of the  
 78705 SCCS file.

78706 **llist** Specify a *list* of releases to which deltas can no longer be made (that is, *get*  
 78707 **-e** against one of these locked releases fails). Conforming applications  
 78708 shall use the following syntax to specify a *list*. Implementations may  
 78709 accept additional forms as an extension:

78710 <list> ::= a | <range-list>  
 78711 <range-list> ::= <range> | <range-list>, <range>  
 78712 <range> ::= <SID>

78713 The character *a* in the *list* shall be equivalent to specifying all releases for  
 78714 the named SCCS file. The non-terminal <SID> in range shall be the delta  
 78715 number of an existing delta associated with the SCCS file.

78716 **n** Cause *delta* to create a null delta in each of those releases (if any) being  
 78717 skipped when a delta is made in a new release (for example, in making  
 78718 delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas  
 78719 shall serve as anchor points so that branch deltas may later be created  
 78720 from them. The absence of this flag shall cause skipped releases to be  
 78721 nonexistent in the SCCS file, preventing branch deltas from being created  
 78722 from them in the future. During the initial creation of an SCCS file, the **n**  
 78723 flag may be ignored; that is, if the **-r** option is used to set the release  
 78724 number of the initial SID to a value greater than 1, null deltas need not be  
 78725 created for the “skipped” releases.

|       |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 78726 | <b>qtext</b>       | Substitute user-definable <i>text</i> for all occurrences of the %Q% keyword in the SCCS file text retrieved by <i>get</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 78727 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78728 | <b>mmod</b>        | Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the <b>m</b> flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' . ' removed.                                                                                                                                                                                                                                                                                                                    |
| 78729 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78730 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78731 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78732 | <b>ttype</b>       | Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 78733 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78734 | <b>vpgm</b>        | Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the <b>m</b> option is also used even if its value is null.)                                                                                                                                                                                                                                                            |
| 78735 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78736 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78737 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78738 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78739 | <b>-d flag</b>     | Remove (delete) the specified <i>flag</i> from an SCCS file. Several <b>-d</b> options may be supplied on a single <i>admin</i> command. See the <b>-f</b> option for allowable <i>flag</i> names. (The <b>l</b> flag gives a <i>list</i> of releases to be unlocked. See the <b>-f</b> option for further description of the <b>l</b> flag and the syntax of a <i>list</i> .)                                                                                                                                                                                                              |
| 78740 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78741 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78742 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78743 | <b>-a login</b>    | Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-a</b> options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a '!', the users so specified shall be denied permission to make deltas. |
| 78744 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78745 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78746 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78747 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78748 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78749 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78750 | <b>-e login</b>    | Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several <b>-e</b> options may be used on a single <i>admin</i> command line.                                                                                                                                                                                                                                                                   |
| 78751 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78752 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78753 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78754 | <b>-y[comment]</b> | Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the <b>-y</b> option shall result in a default comment line being inserted in the form:                                                                                                                                                                                                                                                                                                                               |
| 78755 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78756 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78757 |                    | <code>"date and time created %s %s by %s", &lt;date&gt;, &lt;time&gt;, &lt;login&gt;</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 78758 |                    | where <date> is expressed in the format of the <i>date</i> utility's %Y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.                                                                                                                                                                                                                                                                                                                                  |
| 78759 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78760 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78761 | <b>-m mrlist</b>   | Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the <b>v</b> flag is set and the MR numbers are validated if the <b>v</b> flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the <b>v</b> flag is not set or MR validation fails.                                                                                                                                                      |
| 78762 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78763 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78764 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78765 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78766 | <b>-h</b>          | Check the structure of the SCCS file and compare the newly computed checksum with the checksum that is stored in the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.                                                                                                                                                                                                                                                                                                                                         |
| 78767 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78768 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 78769 |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

78770            **-z**            Recompute the SCCS file checksum and store it in the first line of the SCCS file (see  
78771                            the **-h** option above). Note that use of this option on a truly corrupted file may  
78772                            prevent future detection of the corruption.

### 78773 **OPERANDS**

78774            The following operands shall be supported:

78775            *file*            A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*  
78776                            utility shall behave as though each file in the directory were specified as a named  
78777                            file, except that non-SCCS files (last component of the pathname does not begin  
78778                            with **s**.) and unreadable files shall be silently ignored.

78779            *newfile*        A pathname of an SCCS file to be created.

78780            If exactly one *file* or *newfile* operand appears, and it is **-**, the standard input shall be read; each  
78781                            line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-  
78782                            SCCS files and unreadable files shall be silently ignored.

### 78783 **STDIN**

78784            The standard input shall be a text file used only if **-i** is specified without an option-argument or  
78785                            if a *file* or *newfile* operand is specified as **-**. If the first character of any standard input line is  
78786                            <SOH> in the POSIX locale, the results are unspecified.

### 78787 **INPUT FILES**

78788            The existing SCCS files shall be text files of an unspecified format.

78789            The application shall ensure that the file named by the **-i** option's *name* option-argument shall  
78790                            be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results  
78791                            are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the  
78792                            header for this file shall be 99 999 for this delta.

### 78793 **ENVIRONMENT VARIABLES**

78794            The following environment variables shall affect the execution of *admin*:

78795            **LANG**            Provide a default value for the internationalization variables that are unset or null.  
78796                            (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
78797                            variables used to determine the values of locale categories.)

78798            **LC\_ALL**        If set to a non-empty string value, override the values of all the other  
78799                            internationalization variables.

78800            **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
78801                            characters (for example, single-byte as opposed to multi-byte characters in  
78802                            arguments and input files).

78803            **LC\_MESSAGES**

78804                            Determine the locale that should be used to affect the format and contents of  
78805                            diagnostic messages written to standard error and the contents of the default **-y**  
78806                            comment.

78807            **NLSPATH**     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

### 78808 **ASYNCHRONOUS EVENTS**

78809            Default.

### 78810 **STDOUT**

78811            Not used.



78812 **STDERR**

78813 The standard error shall be used only for diagnostic messages.

78814 **OUTPUT FILES**

78815 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a  
78816 locking *z-file*, as described in *get* (on page 2823), may be created and deleted.

78817 **EXTENDED DESCRIPTION**

78818 None.

78819 **EXIT STATUS**

78820 The following exit values shall be returned:

78821 0 Successful completion.

78822 >0 An error occurred.

78823 **CONSEQUENCES OF ERRORS**

78824 Default.

78825 **APPLICATION USAGE**

78826 It is recommended that directories containing SCCS files be writable by the owner only, and that  
78827 SCCS files themselves be read-only. The mode of the directories should allow only the owner to  
78828 modify SCCS files contained in the directories. The mode of the SCCS files prevents any  
78829 modification at all except by SCCS commands.

78830 **EXAMPLES**

78831 None.

78832 **RATIONALE**

78833 None.

78834 **FUTURE DIRECTIONS**

78835 None.

78836 **SEE ALSO**

78837 *delta, get, prs, what*

78838 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

78839 **CHANGE HISTORY**

78840 First released in Issue 2.

78841 **Issue 6**

78842 The normative text is reworded to avoid use of the term “must” for application requirements,  
78843 and to emphasize the term “shall” for implementation requirements.

78844 The grammar is updated.

78845 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES  
78846 section warning that the maximum lines recorded in the file is 99 999.

78847 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **-h**  
78848 option.

78849 **NAME**

78850 alias ‡define or display aliases

78851 **SYNOPSIS**78852 alias [*alias-name*[=*string*] . . .]78853 **DESCRIPTION**

78854 The *alias* utility shall create or redefine alias definitions or write the values of existing alias  
 78855 definitions to standard output. An alias definition provides a string value that shall replace a  
 78856 command name when it is encountered; see [Section 2.3.1](#) (on page 2348).

78857 An alias definition shall affect the current shell execution environment and the execution  
 78858 environments of the subshells of the current shell. When used as specified by this volume of  
 78859 POSIX.1-2017, the alias definition shall not affect the parent process of the current shell nor any  
 78860 utility environment invoked by the shell; see [Section 2.12](#) (on page 2381).

78861 **OPTIONS**

78862 None.

78863 **OPERANDS**

78864 The following operands shall be supported:

78865 *alias-name* Write the alias definition to standard output.78866 *alias-name=string*78867 Assign the value of *string* to the alias *alias-name*.

78868 If no operands are given, all alias definitions shall be written to standard output.

78869 **STDIN**

78870 Not used.

78871 **INPUT FILES**

78872 None.

78873 **ENVIRONMENT VARIABLES**78874 The following environment variables shall affect the execution of *alias*:

78875 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 78876 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization  
 78877 variables used to determine the values of locale categories.)

78878 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 78879 internationalization variables.

78880 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 78881 characters (for example, single-byte as opposed to multi-byte characters in  
 78882 arguments).

78883 *LC\_MESSAGES*

78884 Determine the locale that should be used to affect the format and contents of  
 78885 diagnostic messages written to standard error.

78886 *XSI NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.78887 **ASYNCHRONOUS EVENTS**

78888 Default.

**78889 STDOUT**

78890 The format for displaying aliases (when no operands or only *name* operands are specified) shall  
78891 be:

```
78892 "%s=%s\n", name, value
```

78893 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the  
78894 shell. See the description of shell quoting in [Section 2.2](#) (on page 2346).

**78895 STDERR**

78896 The standard error shall be used only for diagnostic messages.

**78897 OUTPUT FILES**

78898 None.

**78899 EXTENDED DESCRIPTION**

78900 None.

**78901 EXIT STATUS**

78902 The following exit values shall be returned:

78903 0 Successful completion.

78904 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.

**78905 CONSEQUENCES OF ERRORS**

78906 Default.

**78907 APPLICATION USAGE**

78908 None.

**78909 EXAMPLES**

78910 1. Create a short alias for a commonly used *ls* command:

```
78911 alias lf="ls -CF"
```

78912 2. Create a simple ``redo'' command to repeat previous entries in the command history file:

```
78913 alias r='fc -s'
```

78914 3. Use 1K units for *du*:

```
78915 alias du=du\ -k
```

78916 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

```
78917 alias nohup="nohup "
```

**78918 RATIONALE**

78919 The *alias* description is based on historical KornShell implementations. Known differences exist  
78920 between that and the C shell. The KornShell version was adopted to be consistent with all the  
78921 other KornShell features in this volume of POSIX.1-2017, such as command line editing.

78922 Since *alias* affects the current shell execution environment, it is generally provided as a shell  
78923 regular built-in.

78924 Historical versions of the KornShell have allowed aliases to be exported to scripts that are  
78925 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of  
78926 POSIX.1-2017 only when an explicit extension such as *-x* is used. The standard developers  
78927 considered that aliases were of use primarily to interactive users and that they should normally  
78928 not affect shell scripts called by those users; functions are available to such scripts.

78929 Historical versions of the KornShell had not written aliases in a quoted manner suitable for  
78930 reentry to the shell, but this volume of POSIX.1-2017 has made this a requirement for all similar  
78931 output. Therefore, consistency was chosen over this detail of historical practice.

78932 **FUTURE DIRECTIONS**

78933 None.

78934 **SEE ALSO**

78935 [Section 2.9.5](#) (on page 2374)

78936 XBD [Chapter 8](#) (on page 173)

78937 **CHANGE HISTORY**

78938 First released in Issue 4.

78939 **Issue 6**

78940 This utility is marked as part of the User Portability Utilities option.

78941 The APPLICATION USAGE section is added.

78942 **Issue 7**

78943 The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability  
78944 Utilities is now an option for interactive utilities.

78945 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

78946 The first example is changed to remove the creation of an alias for a standard utility that alters  
78947 its behavior to be non-conforming.

78948 **NAME**

78949 ar — create and maintain library archives

78950 **SYNOPSIS**78951 SD ar -d [-v] *archive file...*78952 XSI ar -m [-v] *archive file...*78953 ar -m -a [-v] *posname archive file...*78954 ar -m -b [-v] *posname archive file...*78955 ar -m -i [-v] *posname archive file...*78956 XSI ar -p [-v] [-s] *archive [file...]*78957 XSI ar -q [-cv] *archive file...*78958 ar -r [-cuv] *archive file...*78959 XSI ar -r -a [-cuv] *posname archive file...*78960 ar -r -b [-cuv] *posname archive file...*78961 ar -r -i [-cuv] *posname archive file...*78962 XSI ar -t [-v] [-s] *archive [file...]*78963 XSI ar -x [-v] [-sCT] *archive [file...]*78964 **DESCRIPTION**78965 The *ar* utility is part of the Software Development Utilities option.

78966 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once  
 78967 an archive has been created, new files can be added, and existing files in an archive can be  
 78968 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the  
 78969 implementation shall format the archive so that it is usable as a library for link editing (see *c99*  
 78970 and *fort77*). When some of the archived files are not valid object files, the suitability of the  
 78971 XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire  
 78972 archive shall be printable.

78973 When *ar* creates an archive, it creates administrative information indicating whether a symbol  
 78974 table is present in the archive. When there is at least one object file that *ar* recognizes as such in  
 78975 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is  
 78976 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update  
 78977 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the  
 78978 symbol table to be rebuilt.

78979 All *file* operands can be pathnames. However, files within archives shall be named by a filename,  
 78980 which is the last component of the pathname used when the file was entered into the archive.  
 78981 The comparison of *file* operands to the names of files in archives shall be performed by  
 78982 comparing the last component of the operand to the name of the file in the archive.

78983 It is unspecified whether multiple files in the archive may be identically named. In the case of  
 78984 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive  
 78985 having a name that is the same as the last component of the operand.

## 78986 OPTIONS

78987 The *ar* utility shall conform to XBD Section 12.2 (on page 216), except for Guideline 9.

78988 The following options shall be supported:

- 78989 XSI **-a** Position new files in the archive after the file named by the *posname* operand.
- 78990 XSI **-b** Position new files in the archive before the file named by the *posname* operand.
- 78991 **-c** Suppress the diagnostic message that is written to standard error by default when the archive *archive* is created.
- 78992
- 78993 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This option is useful when **-T** is also used, to prevent truncated filenames from replacing files with the same prefix.
- 78994
- 78995
- 78996 **-d** Delete one or more *files* from *archive*.
- 78997 XSI **-i** Position new files in the archive before the file in the archive named by the *posname* operand (equivalent to **-b**).
- 78998
- 78999 XSI **-m** Move the named files in the archive. The **-a**, **-b**, or **-i** options with the *posname* operand indicate the position; otherwise, move the names files in the archive to the end of the archive.
- 79000
- 79001
- 79002 **-p** Write the contents of the *files* in the archive named by *file* operands from *archive* to the standard output. If no *file* operands are specified, the contents of all files in the archive shall be written in the order of the archive.
- 79003
- 79004
- 79005 XSI **-q** Append the named files to the end of the archive. In this case *ar* does not check whether the added files are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.
- 79006
- 79007
- 79008 **-r** Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a new archive shall be created and a diagnostic message shall be written to standard error (unless the **-c** option is specified). If no *files* are specified and the *archive* exists, the results are undefined. Files that replace existing files in the archive shall not change the order of the archive. Files that do not replace existing files in the archive shall be appended to the archive unless a **-a**, **-b**, or **-i** option specifies another position.
- 79009
- 79010
- 79011
- 79012
- 79013 XSI
- 79014 XSI **-s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an option that modifies the archive contents. This option is useful to restore the archive symbol table after it has been stripped; see *strip*.
- 79015
- 79016
- 79017 **-t** Write a table of contents of *archive* to the standard output. Only the files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.
- 79018
- 79019
- 79020 XSI **-T** Allow filename truncation of extracted files whose archive names are longer than the file system can support. By default, extracting a file with a name that is too long shall be an error; a diagnostic message shall be written and the file shall not be extracted.
- 79021
- 79022
- 79023
- 79024 **-u** Update older files in the archive. When used with the **-r** option, files in the archive shall be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file in the archive.
- 79025
- 79026

79027           **-v**           Give verbose output. When used with the option characters **-d**, **-r**, or **-x**, write a  
79028 detailed file-by-file description of the archive creation and maintenance activity, as  
79029 described in the STDOUT section.

79030                           When used with **-p**, write the name of the file in the archive to the standard output  
79031 before writing the file in the archive itself to the standard output, as described in  
79032 the STDOUT section.

79033                           When used with **-t**, include a long listing of information about the files in the  
79034 archive, as described in the STDOUT section.

79035           **-x**           Extract the files in the archive named by the *file* operands from *archive*. The  
79036 contents of the archive shall not be changed. If no *file* operands are given, all files  
79037 in the archive shall be extracted. The modification time of each file extracted shall  
79038 be set to the time the file is extracted from the archive.

### 79039 OPERANDS

79040           The following operands shall be supported:

79041           *archive*       A pathname of the archive.

79042           *file*           A pathname. Only the last component shall be used when comparing against the  
79043 names of files in the archive. If two or more *file* operands have the same last  
79044 pathname component (basename), the results are unspecified. The  
79045 implementation's archive format shall not truncate valid filenames of files added  
79046 to or replaced in the archive.

79047 XSI       *posname*       The name of a file in the archive, used for relative positioning; see options **-m** and  
79048 **-r**.

### 79049 STDIN

79050           Not used.

### 79051 INPUT FILES

79052           The archive named by *archive* shall be a file in the format created by *ar -r*.

### 79053 ENVIRONMENT VARIABLES

79054           The following environment variables shall affect the execution of *ar*:

79055           **LANG**           Provide a default value for the internationalization variables that are unset or null.  
79056 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
79057 variables used to determine the values of locale categories.)

79058           **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
79059 internationalization variables.

79060           **LC\_CTYPE**   Determine the locale for the interpretation of sequences of bytes of text data as  
79061 characters (for example, single-byte as opposed to multi-byte characters in  
79062 arguments and input files).

79063           **LC\_MESSAGES**

79064                           Determine the locale that should be used to affect the format and contents of  
79065 diagnostic messages written to standard error.

79066           **LC\_TIME**       Determine the format and content for date and time strings written by *ar -tv*.

79067 XSI       **NLSPATH**       Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

79068 *TMPDIR* Determine the pathname that overrides the default directory for temporary files, if  
79069 any.

79070 *TZ* Determine the timezone used to calculate date and time strings written by *ar -tv*.  
79071 If *TZ* is unset or null, an unspecified default timezone shall be used.

## 79072 ASYNCHRONOUS EVENTS

79073 Default.

## 79074 STDOUT

79075 If the *-d* option is used with the *-v* option, the standard output format shall be:

79076 "d - %s\n", <file>

79077 where *file* is the operand specified on the command line.

79078 If the *-p* option is used with the *-v* option, *ar* shall precede the contents of each file with:

79079 "\n<%s>\n\n", <file>

79080 where *file* is the operand specified on the command line, if *file* operands were specified, and the  
79081 name of the file in the archive if they were not.

79082 If the *-r* option is used with the *-v* option:

79083 If *file* is already in the archive, the standard output format shall be:

79084 "r - %s\n", <file>

79085 where <*file*> is the operand specified on the command line.

79086 If *file* is not already in the archive, the standard output format shall be:

79087 "a - %s\n", <file>

79088 where <*file*> is the operand specified on the command line.

79089 If the *-t* option is used, *ar* shall write the names of the files in the archive to the standard output  
79090 in the format:

79091 "%s\n", <file>

79092 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
79093 name of the file in the archive if they were not.

79094 If the *-t* option is used with the *-v* option, the standard output format shall be:

79095 "%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,  
79096 <group ID>, <number of bytes in member>,  
79097 <abbreviated month>, <day-of-month>, <hour>,  
79098 <minute>, <year>, <file>

79099 where:

79100 <*file*> Shall be the operand specified on the command line, if *file* operands were specified,  
79101 or the name of the file in the archive if they were not.

79102 <*member mode*>

79103 Shall be formatted the same as the <*file mode*> string defined in the STDOUT  
79104 section of *ls*, except that the first character, the <*entry type*>, is not used; the string  
79105 represents the file mode of the file in the archive at the time it was added to or  
79106 replaced in the archive.



79107 The following represent the last-modification time of a file when it was most recently added to  
79108 or replaced in the archive:

79109 <*abbreviated month*>

79110 Equivalent to the format of the %b conversion specification format in *date*.

79111 <*day-of-month*>

79112 Equivalent to the format of the %e conversion specification format in *date*.

79113 <*hour*>

Equivalent to the format of the %H conversion specification format in *date*.

79114 <*minute*>

Equivalent to the format of the %M conversion specification format in *date*.

79115 <*year*>

Equivalent to the format of the %Y conversion specification format in *date*.

79116 When *LC\_TIME* does not specify the POSIX locale, a different format and order of presentation  
79117 of these fields relative to each other may be used in a format appropriate in the specified locale.

79118 If the *-x* option is used with the *-v* option, the standard output format shall be:

79119 "x - %s\n", <*file*>

79120 where *file* is the operand specified on the command line, if *file* operands were specified, or the  
79121 name of the file in the archive if they were not.

#### 79122 **STDERR**

79123 The standard error shall be used only for diagnostic messages. The diagnostic message about  
79124 creating a new archive when *-c* is not specified shall not modify the exit status.

#### 79125 **OUTPUT FILES**

79126 Archives are files with unspecified formats.

#### 79127 **EXTENDED DESCRIPTION**

79128 None.

#### 79129 **EXIT STATUS**

79130 The following exit values shall be returned:

79131 0 Successful completion.

79132 >0 An error occurred.

#### 79133 **CONSEQUENCES OF ERRORS**

79134 Default.

#### 79135 **APPLICATION USAGE**

79136 None.

#### 79137 **EXAMPLES**

79138 None.

#### 79139 **RATIONALE**

79140 The archive format is not described. It is recognized that there are several known *ar* formats,  
79141 which are not compatible. The *ar* utility is included, however, to allow creation of archives that  
79142 are intended for use only on one machine. The archive is specified as a file, and it can be moved  
79143 as a file. This does allow an archive to be moved from one machine to another machine that uses  
79144 the same implementation of *ar*.

79145 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable “archives”. This is a not  
79146 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the  
79147 compilers, based on a historical model.

79148 In historical implementations, the `-q` option (available on XSI-conforming systems) is known to  
79149 execute quickly because *ar* does not check on whether the added members are already in the  
79150 archive. This is useful to bypass the searching otherwise done when creating a large archive  
79151 piece-by-piece. These remarks may but need not remain true for a brand new implementation of  
79152 this utility; hence, these remarks have been moved into the RATIONALE.

79153 BSD implementations historically required applications to provide the `-s` option whenever the  
79154 archive was supposed to contain a symbol table. As in this volume of POSIX.1-2017, System V  
79155 historically creates or updates an archive symbol table whenever an object file is removed from,  
79156 added to, or updated in the archive.

79157 The OPERANDS section requires what might seem to be true without specifying it: the archive  
79158 cannot truncate the filenames below `{NAME_MAX}`. Some historical implementations do so,  
79159 however, causing unexpected results for the application. Therefore, this volume of POSIX.1-2017  
79160 makes the requirement explicit to avoid misunderstandings.

79161 According to the System V documentation, the options `-dmpqrtx` are not required to begin with  
79162 a `<hyphen-minus>` ('-'). This volume of POSIX.1-2017 requires that a conforming application  
79163 use the leading `<hyphen-minus>`.

79164 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as  
79165 an example:

79166 A file created by *ar* begins with the "magic" string `!<arch>\n`". The rest of the archive  
79167 is made up of objects, each of which is composed of a header for a file, a possible filename,  
79168 and the file contents. The header is portable between machine architectures, and, if the file  
79169 contents are printable, the archive is itself printable.

79170 The header is made up of six ASCII fields, followed by a two-character trailer. The fields  
79171 are the object name (16 characters), the file last modification time (12 characters), the user  
79172 and group IDs (each 6 characters), the file mode (8 characters), and the file size (10  
79173 characters). All numeric fields are in decimal, except for the file mode, which is in octal.

79174 The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid`  
79175 and `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field.  
79176 The two-byte trailer is the string `"`<newline>`".

79177 Only the name field has any provision for overflow. If any filename is more than 16  
79178 characters in length or contains an embedded space, the string `"#1/"` followed by the  
79179 ASCII length of the name is written in the name field. The file size (stored in the archive  
79180 header) is incremented by the length of the name. The name is then written immediately  
79181 following the archive header.

79182 Any unused characters in any of these fields are written as `<space>` characters. If any fields  
79183 are their particular maximum number of characters in length, there is no separation  
79184 between the fields.

79185 Objects in the archive are always an even number of bytes long; files that are an odd  
79186 number of bytes long are padded with a `<newline>`, although the size in the header does  
79187 not reflect this.

79188 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an  
79189 object code library, which the linkage editor can use to extract object modules. If the linkage  
79190 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,  
79191 *ar* does not require a symbol table.

79192 The BSD `-o` option was omitted. It is a rare conforming application that uses *ar* to extract object

79193 code from a library with concern for its modification time, since this can only be of importance  
 79194 to *make*. Hence, since this functionality is not deemed important for applications portability, the  
 79195 modification time of the extracted files is set to the current time.

79196 There is at least one known implementation (for a small computer) that can accommodate only  
 79197 object files for that system, disallowing mixed object and other files. The ability to handle any  
 79198 type of file is not only historical practice for most implementations, but is also a reasonable  
 79199 expectation.

79200 Consideration was given to changing the output format of *ar -tv* to the same format as the  
 79201 output of *ls -l*. This would have made parsing the output of *ar* the same as that of *ls*. This was  
 79202 rejected in part because the current *ar* format is commonly used and changes would break  
 79203 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a  
 79204 <slash>. Changing this to be the user name and group name would not be correct if the archive  
 79205 were moved to a machine that contained a different user database. Since *ar* cannot know  
 79206 whether the archive was generated on the same machine, it cannot tell what to report.

79207 The text on the *-ur* option combination is historical practice ‡since one filename can easily  
 79208 represent two different files (for example, */a/foo* and */b/foo*), it is reasonable to replace the file in  
 79209 the archive even when the modification time in the archive is identical to that in the file system.

#### 79210 FUTURE DIRECTIONS

79211 None.

#### 79212 SEE ALSO

79213 *c99, date, fort77, pax, strip*

79214 XBD Chapter 8 (on page 173), Section 12.2 (on page 216), <*unistd.h*>, description of  
 79215 {POSIX\_NO\_TRUNC}

#### 79216 CHANGE HISTORY

79217 First released in Issue 2.

#### 79218 Issue 5

79219 The FUTURE DIRECTIONS section is added.

#### 79220 Issue 6

79221 This utility is marked as part of the Software Development Utilities option.

79222 The STDOUT description is changed for the *-v* option to align with the IEEE P1003.2b draft  
 79223 standard.

79224 The normative text is reworded to avoid use of the term “must” for application requirements.

79225 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

79226 IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use  
 79227 “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being  
 79228 operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is  
 79229 contained in the archive.

79230 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the  
 79231 SYNOPSIS. The change was needed since the *-a*, *-b*, and *-i* options are mutually-exclusive, and  
 79232 *posname* is required if any of these options is specified.

79233 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description  
 79234 of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a  
 79235 backquote followed by a <newline>.

79236 **Issue 7**

79237 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
79238 apply.

79239 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79240 The description of the `-t` option is changed to say “Only the files specified ...”.

79241 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0057 [584] is applied.

79242 **NAME**

79243           asa — interpret carriage-control characters

79244 **SYNOPSIS**79245 FR       asa [*file...*]79246 **DESCRIPTION**79247       The *asa* utility shall write its input files to standard output, mapping carriage-control characters  
79248       from the text files to line-printer control sequences in an implementation-defined manner.79249       The first character of every line shall be removed from the input, and the following actions are  
79250       performed.

79251       If the character removed is:

79252       &lt;space&gt;     The rest of the line is output without change.

79253       0           A &lt;newline&gt; is output, then the rest of the input line.

79254       1           One or more implementation-defined characters that causes an advance to the next  
79255       page shall be output, followed by the rest of the input line.79256       +           The <newline> of the previous line shall be replaced with one or more  
79257       implementation-defined characters that causes printing to return to column  
79258       position 1, followed by the rest of the input line. If the '+' is the first character in  
79259       the input, it shall be equivalent to <space>.79260       The action of the *asa* utility is unspecified upon encountering any character other than those  
79261       listed above as the first character in a line.79262 **OPTIONS**

79263       None.

79264 **OPERANDS**79265       *file*       A pathname of a text file used for input. If no *file* operands are specified, the  
79266       standard input shall be used.79267 **STDIN**79268       The standard input shall be used if no *file* operands are specified, and shall be used if a *file*  
79269       operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,  
79270       the standard input shall not be used. See the INPUT FILES section.79271 **INPUT FILES**

79272       The input files shall be text files.

79273 **ENVIRONMENT VARIABLES**79274       The following environment variables shall affect the execution of *asa*:79275       LANG       Provide a default value for the internationalization variables that are unset or null.  
79276       (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
79277       variables used to determine the values of locale categories.)79278       LC\_ALL      If set to a non-empty string value, override the values of all the other  
79279       internationalization variables.79280       LC\_CTYPE    Determine the locale for the interpretation of sequences of bytes of text data as  
79281       characters (for example, single-byte as opposed to multi-byte characters in  
79282       arguments and input files).

79283 **LC\_MESSAGES**  
 79284 Determine the locale that should be used to affect the format and contents of  
 79285 diagnostic messages written to standard error.

79286 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

79287 **ASYNCHRONOUS EVENTS**  
 79288 Default.

79289 **STDOUT**  
 79290 The standard output shall be the text from the input file modified as described in the  
 79291 DESCRIPTION section.

79292 **STDERR**  
 79293 None.

79294 **OUTPUT FILES**  
 79295 None.

79296 **EXTENDED DESCRIPTION**  
 79297 None.

79298 **EXIT STATUS**  
 79299 The following exit values shall be returned:  
 79300 0 All input files were output successfully.  
 79301 >0 An error occurred.

79302 **CONSEQUENCES OF ERRORS**  
 79303 Default.

79304 **APPLICATION USAGE**  
 79305 None.

79306 **EXAMPLES**

79307 1. The following command:  
 79308 `asa file`  
 79309 permits the viewing of *file* (created by a program using FORTRAN-style carriage-control  
 79310 characters) on a terminal.

79311 2. The following command:  
 79312 `a.out | asa | lp`  
 79313 formats the FORTRAN output of **a.out** and directs it to the printer.

79314 **RATIONALE**  
 79315 The *asa* utility is needed to map “standard” FORTRAN 77 output into a form acceptable to  
 79316 contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.

79317 This utility is generally used only by FORTRAN programs. The standard developers decided to  
 79318 retain *asa* to avoid breaking the historical large base of FORTRAN applications that put carriage-  
 79319 control characters in their output files. There is no requirement that a system have a FORTRAN  
 79320 compiler in order to run applications that need *asa*.

79321 Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII  
 79322 <carriage-return> in response to a '+'. It is suggested that implementations treat characters  
 79323 other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.

79324 However, the action is listed here as ``unspecified'', permitting an implementation to provide  
79325 extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

79326 **FUTURE DIRECTIONS**

79327 None.

79328 **SEE ALSO**

79329 *fort77, lp*

79330 XBD [Chapter 8](#) (on page 173)

79331 **CHANGE HISTORY**

79332 First released in Issue 4.

79333 **Issue 6**

79334 This utility is marked as part of the FORTRAN Runtime Utilities option.

79335 The normative text is reworded to avoid use of the term ``must'' for application requirements.

79336 **Issue 7**

79337 Austin Group Interpretation 1003.1-2001 #092 is applied.

79338 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79339 **NAME**79340 `at` ‡execute commands at a later time79341 **SYNOPSIS**79342 `at` [**-m**] [**-f** *file*] [**-q** *queuename*] **-t** *time\_arg*79343 `at` [**-m**] [**-f** *file*] [**-q** *queuename*] *timespec...*79344 `at` **-r** *at\_job\_id...*79345 `at` **-l** **-q** *queuename*79346 `at` **-l** [*at\_job\_id...*]79347 **DESCRIPTION**79348 The *at* utility shall read commands from standard input and group them together as an *at-job*, to  
79349 be executed at a later time.79350 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process  
79351 group with no controlling terminal, except that the environment variables, current working  
79352 directory, file creation mask, and other implementation-defined execution-time attributes in  
79353 effect when the *at* utility is executed shall be retained and used when the *at-job* is executed.79354 When the *at-job* is submitted, the *at\_job\_id* and scheduled time shall be written to standard error.  
79355 The *at\_job\_id* is an identifier that shall be a string consisting solely of alphanumeric characters  
79356 and the <period> character. The *at\_job\_id* shall be assigned by the system when the job is  
79357 scheduled such that it uniquely identifies a particular job.79358 User notification and the processing of the job's standard output and standard error are  
79359 described under the **-m** option.79360 XSI Users shall be permitted to use *at* if their name appears in the file **at.allow** which is located in an  
79361 implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in  
79362 an implementation-defined directory, shall be checked to determine whether the user shall be  
79363 denied access to *at*. If neither file exists, only a process with appropriate privileges shall be  
79364 allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The  
79365 **at.allow** and **at.deny** files shall consist of one user name per line.79366 **OPTIONS**79367 The *at* utility shall conform to XBD [Section 12.2](#) (on page 216).

79368 The following options shall be supported:

79369 **-f** *file* Specify the pathname of a file to be used as the source of the *at-job*, instead of  
79370 standard input.79371 **-l** (The letter ell.) Report all jobs scheduled for the invoking user if no *at\_job\_id*  
79372 operands are specified. If *at\_job\_ids* are specified, report only information for these  
79373 jobs. The output shall be written to standard output.79374 **-m** Send mail to the invoking user after the *at-job* has run, announcing its completion.  
79375 Standard output and standard error produced by the *at-job* shall be mailed to the  
79376 user as well, unless redirected elsewhere. Mail shall be sent even if the job  
79377 produces no output.79378 If **-m** is not used, the job's standard output and standard error shall be provided to  
79379 the user by means of mail, unless they are redirected elsewhere; if there is no such  
79380 output to provide, the implementation need not notify the user of the job's  
79381 completion.



- 79382        **-q** *queuename*  
 79383                    Specify in which queue to schedule a job for submission. When used with the **-l**  
 79384                    option, limit the search to that particular queue. By default, at-jobs shall be  
 79385                    scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see  
 79386                    *batch*. The meanings of all other *queuenames* are implementation-defined. If **-q** is  
 79387                    specified along with either of the **-t** *time\_arg* or *timespec* arguments, the results are  
 79388                    unspecified.
- 79389        **-r**                    Remove the jobs with the specified *at\_job\_id* operands that were previously  
 79390                    scheduled by the *at* utility.
- 79391        **-t** *time\_arg*        Submit the job to be run at the time specified by the *time* option-argument, which  
 79392                    the application shall ensure has the format as specified by the *touch -t time* utility.

### 79393 OPERANDS

79394        The following operands shall be supported:

- 79395        *at\_job\_id*            The name reported by a previous invocation of the *at* utility at the time the job was  
 79396                    scheduled.
- 79397        *timespec*            Submit the job to be run at the date and time specified. All of the *timespec* operands  
 79398                    are interpreted as if they were separated by <space> characters and concatenated,  
 79399                    and shall be parsed as described in the grammar at the end of this section. The date  
 79400                    and time shall be interpreted as being in the timezone of the user (as determined  
 79401                    by the *TZ* variable), unless a timezone name appears as part of *time*, below.
- 79402                    In the POSIX locale, the following describes the three parts of the time specification  
 79403                    string. All of the values from the *LC\_TIME* categories in the POSIX locale shall be  
 79404                    recognized in a case-insensitive manner.
- 79405        *time*                    The time can be specified as one, two, or four digits. One-digit and  
 79406                    two-digit numbers shall be taken to be hours; four-digit numbers to  
 79407                    be hours and minutes. The time can alternatively be specified as two  
 79408                    numbers separated by a <colon>, meaning *hour:minute*. An AM/PM  
 79409                    indication (one of the values from the **am\_pm** keywords in the  
 79410                    *LC\_TIME* locale category) can follow the time; otherwise, a 24-hour  
 79411                    clock time shall be understood. A timezone name can also follow to  
 79412                    further qualify the time. The acceptable timezone names are  
 79413                    implementation-defined, except that they shall be case-insensitive  
 79414                    and the string **utc** is supported to indicate the time is in Coordinated  
 79415                    Universal Time. In the POSIX locale, the *time* field can also be one of  
 79416                    the following tokens:
- 79417                    **midnight**        Indicates the time 12:00 am (00:00).
- 79418                    **noon**                Indicates the time 12:00 pm.
- 79419                    **now**                Indicates the current day and time. Invoking *at <now>*  
 79420                    shall submit an at-job for potentially immediate  
 79421                    execution (that is, subject only to unspecified  
 79422                    scheduling delays).
- 79423        *date*                    An optional *date* can be specified as either a month name (one of the  
 79424                    values from the **mon** or **abmon** keywords in the *LC\_TIME* locale  
 79425                    category) followed by a day number (and possibly year number  
 79426                    preceded by a comma), or a day of the week (one of the values from  
 79427                    the **day** or **abday** keywords in the *LC\_TIME* locale category). In the

79428 POSIX locale, two special days shall be recognized:

79429 **today** Indicates the current day.

79430 **tomorrow** Indicates the day following the current day.

79431 If no *date* is given, **today** shall be assumed if the given time is greater  
 79432 than the current time, and **tomorrow** shall be assumed if it is less. If  
 79433 the given month is less than the current month (and no year is given),  
 79434 next year shall be assumed.

79435 *increment* The optional *increment* shall be a number preceded by a <plus-sign>  
 79436 ('+') and suffixed by one of the following: **minutes**, **hours**, **days**,  
 79437 **weeks**, **months**, or **years**. (The singular forms shall also be accepted.)  
 79438 The keyword **next** shall be equivalent to an increment number of +1.  
 79439 For example, the following are equivalent commands:

79440 at 2pm + 1 week  
 79441 at 2pm next week

79442 The following grammar describes the precise format of *timespec* in the POSIX locale. The general  
 79443 conventions for this style of grammar are described in [Section 1.3](#) (on page 2335). This formal  
 79444 syntax shall take precedence over the preceding text syntax description. The longest possible  
 79445 token or delimiter shall be recognized at a given point. When used in a *timespec*, white space  
 79446 shall also delimit tokens.

```

79447 %token hr24clock_hr_min
79448 %token hr24clock_hour
79449 /*
79450 An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
79451 or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
79452 may be any of the single digits [0,9], or may be double digits, ranging
79453 from [00,23]. If an hr24clock_hr_min is a four-digit number, the
79454 first two digits shall be a valid hr24clock_hour, while the last two
79455 represent the number of minutes, from [00,59].
79456 */
79457 %token wallclock_hr_min
79458 %token wallclock_hour
79459 /*
79460 A wallclock_hr_min is a one, two-digit, or four-digit number.
79461 A one-digit or two-digit number constitutes a wallclock_hour.
79462 A wallclock_hour may be any of the single digits [1,9], or may
79463 be double digits, ranging from [01,12]. If a wallclock_hr_min
79464 is a four-digit number, the first two digits shall be a valid
79465 wallclock_hour, while the last two represent the number of
79466 minutes, from [00,59].
79467 */
79468 %token minute
79469 /*
79470 A minute is a one or two-digit number whose value can be [0,9]
79471 or [00,59].
79472 */
79473 %token day_number
79474 /*

```

```

79475 A day_number is a number in the range appropriate for the particular
79476 month and year specified by month_name and year_number, respectively.
79477 If no year_number is given, the current year is assumed if the given
79478 date and time are later this year. If no year_number is given and
79479 the date and time have already occurred this year and the month is
79480 not the current month, next year is the assumed year.
79481 */

79482 %token year_number
79483 /*
79484 A year_number is a four-digit number representing the year A.D., in
79485 which the at_job is to be run.
79486 */

79487 %token inc_number
79488 /*
79489 The inc_number is the number of times the succeeding increment
79490 period is to be added to the specified date and time.
79491 */

79492 %token timezone_name
79493 /*
79494 The name of an optional timezone suffix to the time field, in an
79495 implementation-defined format.
79496 */

79497 %token month_name
79498 /*
79499 One of the values from the mon or abmon keywords in the LC_TIME
79500 locale category.
79501 */

79502 %token day_of_week
79503 /*
79504 One of the values from the day or abday keywords in the LC_TIME
79505 locale category.
79506 */

79507 %token am_pm
79508 /*
79509 One of the values from the am_pm keyword in the LC_TIME locale
79510 category.
79511 */

79512 %start timespec
79513 %%
79514 timespec : time
79515 | time date
79516 | time increment
79517 | time date increment
79518 | nowspec
79519 ;

79520 nowspec : "now"
79521 | "now" increment
79522 ;

```

```

79523 time : hr24clock_hr_min
79524 | hr24clock_hr_min timezone_name
79525 | hr24clock_hour ":" minute
79526 | hr24clock_hour ":" minute timezone_name
79527 | wallclock_hr_min am_pm
79528 | wallclock_hr_min am_pm timezone_name
79529 | wallclock_hour ":" minute am_pm
79530 | wallclock_hour ":" minute am_pm timezone_name
79531 | "noon"
79532 | "midnight"
79533 ;

79534 date : month_name day_number
79535 | month_name day_number "," year_number
79536 | day_of_week
79537 | "today"
79538 | "tomorrow"
79539 ;

79540 increment : "+" inc_number inc_period
79541 | "next" inc_period
79542 ;

79543 inc_period : "minute" | "minutes"
79544 | "hour" | "hours"
79545 | "day" | "days"
79546 | "week" | "weeks"
79547 | "month" | "months"
79548 | "year" | "years"
79549 ;

```

## 79550 STDIN

79551 The standard input shall be a text file consisting of commands acceptable to the shell command  
 79552 language described in [Chapter 2](#) (on page 2345). The standard input shall only be used if no `-f`  
 79553 *file* option is specified.

## 79554 INPUT FILES

79555 See the STDIN section.

79556 XSI The text files `at.allow` and `at.deny`, which are located in an implementation-defined directory,  
 79557 shall contain zero or more user names, one per line, of users who are, respectively, authorized or  
 79558 denied access to the *at* and *batch* utilities.

## 79559 ENVIRONMENT VARIABLES

79560 The following environment variables shall affect the execution of *at*:

79561 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 79562 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 79563 variables used to determine the values of locale categories.)

79564 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 79565 internationalization variables.

79566 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 79567 characters (for example, single-byte as opposed to multi-byte characters in  
 79568 arguments and input files).

79569 *LC\_MESSAGES*  
 79570 Determine the locale that should be used to affect the format and contents of  
 79571 diagnostic messages written to standard error and informative messages written to  
 79572 standard output.

79573 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

79574 *LC\_TIME* Determine the format and contents for date and time strings written and accepted  
 79575 by *at*.

79576 *SHELL* Determine a name of a command interpreter to be used to invoke the *at*-job. If the  
 79577 variable is unset or null, *sh* shall be used. If it is set to a value other than a name for  
 79578 *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the  
 79579 login shell from the user database; or any of the preceding accompanied by a  
 79580 warning diagnostic about which was chosen.

79581 *TZ* Determine the timezone. The job shall be submitted for execution at the time  
 79582 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*  
 79583 variable. If *timespec* specifies a timezone, it shall override *TZ*. If *timespec* does not  
 79584 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall  
 79585 be used.

79586 **ASYNCHRONOUS EVENTS**  
 79587 Default.

79588 **STDOUT**  
 79589 When standard input is a terminal, prompts of unspecified format for each line of the user input  
 79590 described in the STDIN section may be written to standard output.

79591 In the POSIX locale, the following shall be written to the standard output for each job when jobs  
 79592 are listed in response to the *-l* option:

79593 "%s\t%s\n", *at\_job\_id*, <*date*>  
 79594 where *date* shall be equivalent in format to the output of:  
 79595 date +"%a %b %e %T %Y"  
 79596 The date and time written shall be adjusted so that they appear in the timezone of the user (as  
 79597 determined by the *TZ* variable).

79598 **STDERR**  
 79599 In the POSIX locale, the following shall be written to standard error when a job has been  
 79600 successfully submitted:

79601 "job %s at %s\n", *at\_job\_id*, <*date*>  
 79602 where *date* has the same format as that described in the STDOUT section. Neither this, nor  
 79603 warning messages concerning the selection of the command interpreter, shall be considered a  
 79604 diagnostic that changes the exit status.

79605 Diagnostic messages, if any, shall be written to standard error.

79606 **OUTPUT FILES**  
 79607 None.

79608 **EXTENDED DESCRIPTION**  
 79609 None.

79610 **EXIT STATUS**

79611 The following exit values shall be returned:

79612 0 The *at* utility successfully submitted, removed, or listed a job or jobs.

79613 &gt;0 An error occurred.

79614 **CONSEQUENCES OF ERRORS**

79615 The job shall not be scheduled, removed, or listed.

79616 **APPLICATION USAGE**79617 The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other  
79618 cultures may be supported with substantially different interfaces, although implementations are  
79619 encouraged to provide comparable levels of functionality.79620 Since the commands run in a separate shell invocation, running in a separate process group with  
79621 no controlling terminal, open file descriptors, traps, and priority inherited from the invoking  
79622 environment are lost.79623 Some implementations do not allow substitution of different shells using *SHELL*. System V  
79624 systems, for example, have used the login shell value for the user in */etc/passwd*. To select  
79625 reliably another command interpreter, the user must include it as part of the script, such as:79626 \$ at 1800  
79627 myshell myscript  
79628 EOT  
79629 **job ... at ...**  
79630 \$79631 **EXAMPLES**

79632 1. This sequence can be used at a terminal:

79633 at -m 0730 tomorrow  
79634 sort < file >outfile  
79635 EOT79636 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
79637 command procedure (the sequence of output redirection specifications is significant):79638 at now + 1 hour <<!  
79639 diff file1 file2 2>&1 >outfile | mailx mygroup  
79640 !79641 3. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this  
79642 daily processing script named **my.daily** runs every day (although *crontab* is a more  
79643 appropriate vehicle for such work):79644 # my.daily runs every day  
79645 *daily processing*  
79646 at now tomorrow < my.daily79647 4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as  
79648 there are no ambiguities. Examples of various times and operand presentation include:79649 at 0815am Jan 24  
79650 at 8 :15amjan24  
79651 at now "+ 1day"  
79652 at 5 pm FRIday  
79653 at '17

```
79654 utc+
79655 30minutes'
```

## 79656 RATIONALE

79657 The *at* utility reads from standard input the commands to be executed at a later time. It may be  
79658 useful to redirect standard output and standard error within the specified commands.

79659 The *-t time* option was added as a new capability to support an internationalized way of  
79660 specifying a time for execution of the submitted job.

79661 Early proposals added a “jobname” concept as a way of giving submitted jobs names that are  
79662 meaningful to the user submitting them. The historical, system-specified *at\_job\_id* gives no  
79663 indication of what the job is. Upon further reflection, it was decided that the benefit of this was  
79664 not worth the change in historical interface. The *at* functionality is useful in simple  
79665 environments, but in large or complex situations, the functionality provided by the Batch  
79666 Services option is more suitable.

79667 The *-q* option historically has been an undocumented option, used mainly by the *batch* utility.

79668 The System V *-m* option was added to provide a method for informing users that an at-job had  
79669 completed. Otherwise, users are only informed when output to standard error or standard  
79670 output are not redirected.

79671 The behavior of *at <now>* was changed in an early proposal from being unspecified to  
79672 submitting a job for potentially immediate execution. Historical BSD *at* implementations support  
79673 this. Historical System V implementations give an error in that case, but a change to the System  
79674 V versions should have no backwards-compatibility ramifications.

79675 On BSD-based systems, a *-u user* option has allowed those with appropriate privileges to access  
79676 the work of other users. Since this is primarily a system administration feature and is not  
79677 universally implemented, it has been omitted. Similarly, a specification for the output format for  
79678 a user with appropriate privileges viewing the queues of other users has been omitted.

79679 The *-f file* option from System V is used instead of the BSD method of using the last operand as  
79680 the pathname. The BSD method is ambiguous ‡does:

```
79681 at 1200 friday
```

79682 mean the same thing if there is a file named **friday** in the current directory?

79683 The *at\_job\_id* is composed of a limited character set in historical practice, and it is mandated here  
79684 to invalidate systems that might try using characters that require shell quoting or that could not  
79685 be easily parsed by shell scripts.

79686 The *at* utility varies between System V and BSD systems in the way timezones are used. On  
79687 System V systems, the *TZ* variable affects the at-job submission times and the times displayed  
79688 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved  
79689 with the current specification. If the user wishes to have the timezone default to that of the  
79690 system, they merely need to issue the *at* command immediately following an unsetting or null  
79691 assignment to *TZ*. For example:

```
79692 TZ= at noon ...
```

79693 gives the desired BSD result.

79694 While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with  
79695 respect to the digit strings, a lexical analyzer would probably be written to look for and return  
79696 digit strings in those cases. The parser could then check whether the digit string returned is a  
79697 valid *day\_number*, *year\_number*, and so on, based on the context.

79698 **FUTURE DIRECTIONS**

79699 None.

79700 **SEE ALSO**79701 *batch, crontab*79702 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)79703 **CHANGE HISTORY**

79704 First released in Issue 2.

79705 **Issue 6**

79706 This utility is marked as part of the User Portability Utilities option.

79707 The following new requirements on POSIX implementations derive from alignment with the  
79708 Single UNIX Specification:79709 If **-m** is not used, the job's standard output and standard error are provided to the user by  
79710 mail.79711 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are  
79712 specified.

79713 The normative text is reworded to avoid use of the term ``must'' for application requirements.

79714 **Issue 7**79715 The *at* utility is moved from the User Portability Utilities option to the Base. User Portability  
79716 Utilities is now an option for interactive utilities.79717 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
79718 by the *at* utility.

79719 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



79720 **NAME**

79721 awk — pattern scanning and processing language

79722 **SYNOPSIS**79723 awk [-F *sepstring*] [-v *assignment*]... *program* [*argument*...]79724 awk [-F *sepstring*] -f *progfile* [-f *progfile*]... [-v *assignment*]...79725 [*argument*...]79726 **DESCRIPTION**

79727 The *awk* utility shall execute programs written in the *awk* programming language, which is  
 79728 specialized for textual data manipulation. An *awk* program is a sequence of patterns and  
 79729 corresponding actions. When input is read that matches a pattern, the action associated with that  
 79730 pattern is carried out.

79731 Input shall be interpreted as a sequence of records. By default, a record is a line, less its  
 79732 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of  
 79733 input shall be matched in turn against each pattern in the program. For each pattern matched,  
 79734 the associated action shall be executed.

79735 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field  
 79736 is a string of non-<blank> non-<newline> characters. This default <blank> and <newline> field  
 79737 delimiter can be changed by using the **FS** built-in variable or the **-F *sepstring*** option. The *awk*  
 79738 utility shall denote the first field in a record \$1, the second \$2, and so on. The symbol \$0 shall  
 79739 refer to the entire record; setting any other field causes the re-evaluation of \$0. Assigning to \$0  
 79740 shall reset the values of all other fields and the **NF** built-in variable.

79741 **OPTIONS**79742 The *awk* utility shall conform to XBD [Section 12.2](#) (on page 216).

79743 The following options shall be supported:

79744 **-F *sepstring*** Define the input field separator. This option shall be equivalent to:79745 **-v *FS=sepstring***

79746 except that if **-F *sepstring*** and **-v *FS=sepstring*** are both used, it is unspecified  
 79747 whether the **FS** assignment resulting from **-F *sepstring*** is processed in command  
 79748 line order or is processed after the last **-v *FS=sepstring***. See the description of the  
 79749 **FS** built-in variable, and how it is used, in the EXTENDED DESCRIPTION section.

79750 **-f *progfile*** Specify the pathname of the file *progfile* containing an *awk* program. A pathname of  
 79751 '-' shall denote the standard input. If multiple instances of this option are  
 79752 specified, the concatenation of the files specified as *progfile* in the order specified  
 79753 shall be the *awk* program. The *awk* program can alternatively be specified in the  
 79754 command line as a single argument.

79755 **-v *assignment***

79756 The application shall ensure that the *assignment* argument is in the same form as an  
 79757 *assignment* operand. The specified variable assignment shall occur prior to  
 79758 executing the *awk* program, including the actions associated with **BEGIN** patterns  
 79759 (if any). Multiple occurrences of this option can be specified.

79760 **OPERANDS**

79761 The following operands shall be supported:

79762 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*  
 79763 program. The application shall supply the *program* operand as a single argument to  
 79764 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

79765 *argument* Either of the following two types of *argument* can be intermixed:

79766 *file* A pathname of a file that contains the input to be read, which is  
79767 matched against the set of patterns in the program. If no *file* operands  
79768 are specified, or if a *file* operand is '-', the standard input shall be  
79769 used.

79770 *assignment* An operand that begins with an <underscore> or alphabetic  
79771 character from the portable character set (see the table in XBD [Section](#)  
79772 [6.1](#), on page 125), followed by a sequence of underscores, digits, and  
79773 alphabetic characters from the portable character set, followed by the '='  
79774 character, shall specify a variable assignment rather than a pathname.  
79775 The characters before the '=' represent the name of an *awk* variable;  
79776 if that name is an *awk* reserved word (see [Grammar](#), on page 2500)  
79777 the behavior is undefined. The characters following the <equals-  
79778 sign> shall be interpreted as if they appeared in the *awk* program  
79779 preceded and followed by a double-quote ('"') character, as a  
79780 **STRING** token (see [Grammar](#), on page 2500), except that if the last  
79781 character is an unescaped <backslash>, it shall be interpreted as a  
79782 literal <backslash> rather than as the first character of the sequence  
79783 "\\". The variable shall be assigned the value of that **STRING**  
79784 token and, if appropriate, shall be considered a *numeric string* (see  
79785 [Expressions in awk](#), on page 2485), the variable shall also be assigned  
79786 its numeric value. Each such variable assignment shall occur just  
79787 prior to the processing of the following *file*, if any. Thus, an  
79788 assignment before the first *file* argument shall be executed after the  
79789 **BEGIN** actions (if any), while an assignment after the last *file*  
79790 argument shall occur before the **END** actions (if any). If there are no  
79791 *file* arguments, assignments shall be executed before processing the  
79792 standard input.

### 79793 **STDIN**

79794 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-',  
79795 or if a *progfile* option-argument is '-'; see the INPUT FILES section. If the *awk* program contains  
79796 no actions and no patterns, but is otherwise a valid *awk* program, standard input and any *file*  
79797 operands shall not be read and *awk* shall exit with a return status of zero.

### 79798 **INPUT FILES**

79799 Input files to the *awk* program from any of the following sources shall be text files:

79800 Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV**  
79801 and **ARGC**

79802 Standard input in the absence of any *file* operands

79803 Arguments to the **getline** function

79804 Whether the variable **RS** is set to a value other than a <newline> or not, for these files,  
79805 implementations shall support records terminated with the specified separator up to  
79806 {LINE\_MAX} bytes and may support longer records.

79807 If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile*  
79808 option-arguments are text files and their concatenation, in the same order as they appear in the  
79809 arguments, is an *awk* program.

79810 **ENVIRONMENT VARIABLES**79811 The following environment variables shall affect the execution of *awk*:

79812 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 79813 (See XBD Section 8.2 (on page 174) for the precedence of internationalization  
 79814 variables used to determine the values of locale categories.)

79815 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 79816 internationalization variables.

79817 *LC\_COLLATE*

79818 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
 79819 character collating elements within regular expressions and in comparisons of  
 79820 string values.

79821 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 79822 characters (for example, single-byte as opposed to multi-byte characters in  
 79823 arguments and input files), the behavior of character classes within regular  
 79824 expressions, the identification of characters as letters, and the mapping of  
 79825 uppercase and lowercase characters for the **toupper** and **tolower** functions.

79826 *LC\_MESSAGES*

79827 Determine the locale that should be used to affect the format and contents of  
 79828 diagnostic messages written to standard error.

79829 *LC\_NUMERIC*

79830 Determine the radix character used when interpreting numeric input, performing  
 79831 conversions between numeric and string values, and formatting numeric output.  
 79832 Regardless of locale, the <period> character (the decimal-point character of the  
 79833 POSIX locale) is the decimal-point character recognized in processing *awk*  
 79834 programs (including assignments in command line arguments).

79835 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

79836 *PATH* Determine the search path when looking for commands executed by *system(expr)*,  
 79837 or input and output pipes; see XBD Chapter 8 (on page 173).

79838 In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.79839 **ASYNCHRONOUS EVENTS**

79840 Default.

79841 **STDOUT**79842 The nature of the output files depends on the *awk* program.79843 **STDERR**

79844 The standard error shall be used only for diagnostic messages.

79845 **OUTPUT FILES**79846 The nature of the output files depends on the *awk* program.

79847 **EXTENDED DESCRIPTION**79848 **Overall Program Structure**

79849 An *awk* program is composed of pairs of the form:

79850 `pattern { action }`

79851 Either the pattern or the action (including the enclosing brace characters) can be omitted.

79852 A missing pattern shall match any record of input, and a missing action shall be equivalent to:

79853 `{ print }`

79854 Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN**  
 79855 patterns in the order they occur in the program. Then each *file* operand (or standard input if no  
 79856 files were specified) shall be processed in turn by reading data from the file until a record  
 79857 separator is seen (<newline> by default). Before the first reference to a field in the record is  
 79858 evaluated, the record shall be split into fields, according to the rules in [Regular Expressions](#) (on  
 79859 page 2491), using the value of **FS** that was current at the time the record was read. Each pattern  
 79860 in the program then shall be evaluated in the order of occurrence, and the action associated with  
 79861 each pattern that matches the current record executed. The action for a matching pattern shall be  
 79862 executed before evaluating subsequent patterns. Finally, the actions associated with all **END**  
 79863 patterns shall be executed in the order they occur in the program.

79864 **Expressions in awk**

79865 Expressions describe computations used in *patterns* and *actions*. In the following table, valid  
 79866 expression operations are given in groups from highest precedence first to lowest precedence  
 79867 last, with equal-precedence operators grouped between horizontal lines. In expression  
 79868 evaluation, where the grammar is formally ambiguous, higher precedence operators shall be  
 79869 evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent  
 79870 any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side  
 79871 of an assignment operator). The precise syntax of expressions is given in [Grammar](#) (on page  
 79872 2500).

79873 **Table 4-1** Expressions in Decreasing Precedence in *awk*

| 79874 | Syntax                   | Name            | Type of Result      | Associativity |
|-------|--------------------------|-----------------|---------------------|---------------|
| 79875 | <code>( expr )</code>    | Grouping        | Type of <i>expr</i> | N/A           |
| 79876 | <code>\$expr</code>      | Field reference | String              | N/A           |
| 79877 | <code>lvalue ++</code>   | Post-increment  | Numeric             | N/A           |
| 79878 | <code>lvalue --</code>   | Post-decrement  | Numeric             | N/A           |
| 79879 | <code>++ lvalue</code>   | Pre-increment   | Numeric             | N/A           |
| 79880 | <code>-- lvalue</code>   | Pre-decrement   | Numeric             | N/A           |
| 79881 | <code>expr ^ expr</code> | Exponentiation  | Numeric             | Right         |
| 79882 | <code>! expr</code>      | Logical not     | Numeric             | N/A           |
| 79883 | <code>+ expr</code>      | Unary plus      | Numeric             | N/A           |
| 79884 | <code>- expr</code>      | Unary minus     | Numeric             | N/A           |
| 79885 | <code>expr * expr</code> | Multiplication  | Numeric             | Left          |
| 79886 | <code>expr / expr</code> | Division        | Numeric             | Left          |
| 79887 | <code>expr % expr</code> | Modulus         | Numeric             | Left          |

| Syntax                                     | Name                             | Type of Result                                   | Associativity |
|--------------------------------------------|----------------------------------|--------------------------------------------------|---------------|
| <i>expr</i> + <i>expr</i>                  | Addition                         | Numeric                                          | Left          |
| <i>expr</i> - <i>expr</i>                  | Subtraction                      | Numeric                                          | Left          |
| <i>expr</i> <i>expr</i>                    | String concatenation             | String                                           | Left          |
| <i>expr</i> < <i>expr</i>                  | Less than                        | Numeric                                          | None          |
| <i>expr</i> <= <i>expr</i>                 | Less than or equal to            | Numeric                                          | None          |
| <i>expr</i> != <i>expr</i>                 | Not equal to                     | Numeric                                          | None          |
| <i>expr</i> == <i>expr</i>                 | Equal to                         | Numeric                                          | None          |
| <i>expr</i> > <i>expr</i>                  | Greater than                     | Numeric                                          | None          |
| <i>expr</i> >= <i>expr</i>                 | Greater than or equal to         | Numeric                                          | None          |
| <i>expr</i> ~ <i>expr</i>                  | ERE match                        | Numeric                                          | None          |
| <i>expr</i> !~ <i>expr</i>                 | ERE non-match                    | Numeric                                          | None          |
| <i>expr</i> in array                       | Array membership                 | Numeric                                          | Left          |
| ( <i>index</i> ) in array                  | Multi-dimension array membership | Numeric                                          | Left          |
| <i>expr</i> && <i>expr</i>                 | Logical AND                      | Numeric                                          | Left          |
| <i>expr</i>    <i>expr</i>                 | Logical OR                       | Numeric                                          | Left          |
| <i>expr1</i> ? <i>expr2</i> : <i>expr3</i> | Conditional expression           | Type of selected<br><i>expr2</i> or <i>expr3</i> | Right         |
| <i>lvalue</i> ^= <i>expr</i>               | Exponentiation assignment        | Numeric                                          | Right         |
| <i>lvalue</i> %= <i>expr</i>               | Modulus assignment               | Numeric                                          | Right         |
| <i>lvalue</i> *= <i>expr</i>               | Multiplication assignment        | Numeric                                          | Right         |
| <i>lvalue</i> /= <i>expr</i>               | Division assignment              | Numeric                                          | Right         |
| <i>lvalue</i> += <i>expr</i>               | Addition assignment              | Numeric                                          | Right         |
| <i>lvalue</i> -= <i>expr</i>               | Subtraction assignment           | Numeric                                          | Right         |
| <i>lvalue</i> = <i>expr</i>                | Assignment                       | Type of <i>expr</i>                              | Right         |

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value either by the equivalent of the following calls to functions defined by the ISO C standard:

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

or by converting the initial portion of the string to type **double** representation as follows:

The input string is decomposed into two parts: an initial, possibly empty, sequence of white-space characters (as specified by *isspace()*) and a subject sequence interpreted as a floating-point constant.

The expected form of the subject sequence is an optional '+' or '-' sign, then a non-empty sequence of digits optionally containing a <period>, then an optional exponent part. An exponent part consists of 'e' or 'E', followed by an optional sign, followed by one or more decimal digits.

The sequence starting with the first digit or the <period> (whichever occurs first) is interpreted as a floating constant of the C language, and if neither an exponent part nor a <period> appears, a <period> is assumed to follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the value resulting from the conversion is negated.

A numeric value that is exactly equal to the value of an integer (see [Section 1.1.2](#), on page 2331)

79934 shall be converted to a string by the equivalent of a call to the **sprintf** function (see [String](#)  
 79935 [Functions](#), on page 2497) with the string "%d" as the *fmt* argument and the numeric value being  
 79936 converted as the first and only *expr* argument. Any other numeric value shall be converted to a  
 79937 string by the equivalent of a call to the **sprintf** function with the value of the variable  
 79938 **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only  
 79939 *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a  
 79940 floating-point format specification. This volume of POSIX.1-2017 specifies no explicit  
 79941 conversions between numbers and strings. An application can force an expression to be treated  
 79942 as a number by adding zero to it, or can force it to be treated as a string by concatenating the null  
 79943 string (" ") to it.

79944 A string value shall be considered a *numeric string* if it comes from one of the following:

- 79945 1. Field variables
- 79946 2. Input from the *getline()* function
- 79947 3. **FILENAME**
- 79948 4. **ARGV** array elements
- 79949 5. **ENVIRON** array elements
- 79950 6. Array elements created by the *split()* function
- 79951 7. A command line variable assignment
- 79952 8. Variable assignment from another numeric string variable

79953 and an implementation-dependent condition corresponding to either case (a) or (b) below is  
 79954 met.

- 79955 a. After the equivalent of the following calls to functions defined by the ISO C standard,  
 79956 *string\_value\_end* would differ from *string\_value*, and any characters before the terminating  
 79957 null character in *string\_value\_end* would be <blank> characters:

```
79958 char *string_value_end;
79959 setlocale(LC_NUMERIC, "");
79960 numeric_value = strtod (string_value, &string_value_end);
```

- 79961 b. After all the following conversions have been applied, the resulting string would lexically  
 79962 be recognized as a **NUMBER** token as described by the lexical conventions in [Grammar](#)  
 79963 (on page 2500):

79964 † If leading and trailing <blank> characters are discarded.

79965 † If the first non-<blank> is '+' or '-', it is discarded.

79966 † If occurrence of the decimal point character from the current locale is changed to  
 79967 a <period>.

79968 In case (a) the numeric value of the *numeric string* shall be the value that would be returned by  
 79969 the *strtod()* call. In case (b) if the first non-<blank> is '-', the numeric value of the *numeric string*  
 79970 shall be the negation of the numeric value of the recognized **NUMBER** token; otherwise, the  
 79971 numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER**  
 79972 token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that  
 79973 term is used in this section.

79974 When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall  
 79975 be treated as false and any other value shall be treated as true. Otherwise, a string value of the  
 79976 null string shall be treated as false and any other value shall be treated as true. A Boolean

79977 context shall be one of the following:

79978       The first subexpression of a conditional expression

79979       An expression operated on by logical NOT, logical AND, or logical OR

79980       The second expression of a **for** statement

79981       The expression of an **if** statement

79982       The expression of the **while** clause in either a **while** or **do...while** statement

79983       An expression used as a pattern (as in Overall Program Structure)

79984 All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C

79985 standard (see [Section 1.1.2](#), on page 2331).

79986 The value of the expression:

79987 `expr1 ^ expr2`

79988 shall be equivalent to the value returned by the ISO C standard function call:

79989 `pow(expr1, expr2)`

79990 The expression:

79991 `lvalue ^= expr`

79992 shall be equivalent to the ISO C standard expression:

79993 `lvalue = pow(lvalue, expr)`

79994 except that `lvalue` shall be evaluated only once. The value of the expression:

79995 `expr1 % expr2`

79996 shall be equivalent to the value returned by the ISO C standard function call:

79997 `fmod(expr1, expr2)`

79998 The expression:

79999 `lvalue %= expr`

80000 shall be equivalent to the ISO C standard expression:

80001 `lvalue = fmod(lvalue, expr)`

80002 except that `lvalue` shall be evaluated only once.

80003 Variables and fields shall be set by the assignment statement:

80004 `lvalue = expression`

80005 and the type of *expression* shall determine the resulting variable type. The assignment includes

80006 the arithmetic assignments ("`+=`", "`-=`", "`*=`", "`/=`", "`%=`", "`^=`", "`++`", "`--`") all of which

80007 shall produce a numeric result. The left-hand side of an assignment and the target of increment

80008 and decrement operators can be one of a variable, an array with index, or a field selector.

80009 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not

80010 be declared. They shall initially be empty, and their sizes shall change dynamically. The

80011 subscripts, or element identifiers, are strings, providing a type of associative array capability. An

80012 array name followed by a subscript within square brackets can be used as an lvalue and thus as

80013 an expression, as described in the grammar; see [Grammar](#) (on page 2500). Unsubscripted array

80014 names can be used in only the following contexts:

80015 A parameter in a function definition or function call  
 80016 The **NAME** token following any use of the keyword **in** as specified in the grammar (see  
 80017 [Grammar](#), on page 2500); if the name used in this context is not an array name, the  
 80018 behavior is undefined

80019 A valid array *index* shall consist of one or more <comma>-separated expressions, similar to the  
 80020 way in which multi-dimensional arrays are indexed in some programming languages. Because  
 80021 *awk* arrays are really one-dimensional, such a <comma>-separated list shall be converted to a  
 80022 single string by concatenating the string values of the separate expressions, each separated from  
 80023 the other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be  
 80024 equivalent:

```
80025 var[expr1, expr2, ... exprn]
```

```
80026 var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

80027 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is  
 80028 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall  
 80029 not cause that element to exist. Any other reference to a nonexistent array element shall  
 80030 automatically create it.

80031 Comparisons (with the '<', '<=', '!=', '==', '>', and '>=' operators) shall be made  
 80032 numerically if both operands are numeric, if one is numeric and the other has a string value that  
 80033 is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise,  
 80034 operands shall be converted to strings as required and a string comparison shall be made as  
 80035 follows:

80036 For the '!=' and '==' operators, the strings should be compared to check if they are  
 80037 identical but may be compared using the locale-specific collation sequence to check if they  
 80038 collate equally.

80039 For the other operators, the strings shall be compared using the locale-specific collation  
 80040 sequence.

80041 The value of the comparison expression shall be 1 if the relation is true, or 0 if the relation is  
 80042 false.

### 80043 **Variables and Special Variables**

80044 Variables can be used in an *awk* program by referencing them. With the exception of function  
 80045 parameters (see [User-Defined Functions](#), on page 2499), they are not explicitly declared.  
 80046 Function parameter names shall be local to the function; all other variable names shall be global.  
 80047 The same name shall not be used as both a function parameter name and as the name of a  
 80048 function or a special *awk* variable. The same name shall not be used both as a variable name with  
 80049 global scope and as the name of a function. The same name shall not be used within the same  
 80050 scope both as a scalar variable and as an array. Uninitialized variables, including scalar  
 80051 variables, array elements, and field variables, shall have an uninitialized value. An uninitialized  
 80052 value shall have both a numeric value of zero and a string value of the empty string. Evaluation  
 80053 of variables with an uninitialized value, to either string or numeric, shall be determined by the  
 80054 context in which they are used.

80055 Field variables shall be designated by a '\$' followed by a number or numerical expression. The  
 80056 effect of the field number *expression* evaluating to anything other than a non-negative integer is  
 80057 unspecified; uninitialized variables or string values need not be converted to numeric values in  
 80058 this context. New field variables can be created by assigning a value to them. References to  
 80059 nonexistent fields (that is, fields after **\$NF**), shall evaluate to the uninitialized value. Such



80060 references shall not create new fields. However, assigning to a nonexistent field (for example,  
80061  $\$(NF+2)=5$ ) shall increase the value of **NF**; create any intervening fields with the uninitialized  
80062 value; and cause the value of  $\$0$  to be recomputed, with the fields being separated by the value  
80063 of **OFS**. Each field variable shall have a string value or an uninitialized value when created.  
80064 Field variables shall have the uninitialized value when created from  $\$0$  using **FS** and the variable  
80065 does not contain any characters. If appropriate, the field variable shall be considered a numeric  
80066 string (see [Expressions in awk](#), on page 2485).

80067 Implementations shall support the following other special variables that are set by *awk*:

80068 **ARGC** The number of elements in the **ARGV** array.

80069 **ARGV** An array of command line arguments, excluding options and the *program*  
80070 argument, numbered from zero to **ARGC**-1.

80071 The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As  
80072 each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the  
80073 current value of **ARGC**-1, inclusive, as the name of the next input file. Thus,  
80074 setting an element of **ARGV** to null means that it shall not be treated as an input  
80075 file. The name '-' indicates the standard input. If an argument matches the format  
80076 of an *assignment* operand, this argument shall be treated as an *assignment* rather  
80077 than a *file* argument.

80078 **CONVFMT** The **printf** format for converting numbers to strings (except for output statements,  
80079 where **OFMT** is used); "% . 6g" by default.

80080 **ENVIRON** An array representing the value of the environment, as described in the *exec*  
80081 functions defined in the System Interfaces volume of POSIX.1-2017. The indices of  
80082 the array shall be strings consisting of the names of the environment variables, and  
80083 the value of each array element shall be a string consisting of the value of that  
80084 variable. If appropriate, the environment variable shall be considered a *numeric*  
80085 *string* (see [Expressions in awk](#), on page 2485); the array element shall also have its  
80086 numeric value.

80087 In all cases where the behavior of *awk* is affected by environment variables  
80088 (including the environment of any commands that *awk* executes via the **system**  
80089 function or via pipeline redirections with the **print** statement, the **printf** statement,  
80090 or the **getline** function), the environment used shall be the environment at the time  
80091 *awk* began executing; it is implementation-defined whether any modification of  
80092 **ENVIRON** affects this environment.

80093 **FILENAME** A pathname of the current input file. Inside a **BEGIN** action the value is  
80094 undefined. Inside an **END** action the value shall be the name of the last input file  
80095 processed.

80096 **FNR** The ordinal number of the current record in the current file. Inside a **BEGIN** action  
80097 the value shall be zero. Inside an **END** action the value shall be the number of the  
80098 last record processed in the last file processed.

80099 **FS** Input field separator regular expression; a <space> by default.

80100 **NF** The number of fields in the current record. Inside a **BEGIN** action, the use of **NF** is  
80101 undefined unless a **getline** function without a *var* argument is executed previously.  
80102 Inside an **END** action, **NF** shall retain the value it had for the last record read,  
80103 unless a subsequent, redirected, **getline** function without a *var* argument is  
80104 performed prior to entering the **END** action.

|       |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 80105 | <b>NR</b>      | The ordinal number of the current record from the start of input. Inside a <b>BEGIN</b> action the value shall be zero. Inside an <b>END</b> action the value shall be the number of the last record processed.                                                                                                                                                                                                                                                                                                          |
| 80106 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80107 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80108 | <b>OFMT</b>    | The <b>printf</b> format for converting numbers to strings in output statements (see <a href="#">Output Statements</a> , on page 2495); "% . 6g" by default. The result of the conversion is unspecified if the value of <b>OFMT</b> is not a floating-point format specification.                                                                                                                                                                                                                                       |
| 80109 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80110 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80111 | <b>OFS</b>     | The <b>print</b> statement output field separator; <space> by default.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 80112 | <b>ORS</b>     | The <b>print</b> statement output record separator; a <newline> by default.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 80113 | <b>RLENGTH</b> | The length of the string matched by the <b>match</b> function.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 80114 | <b>RS</b>      | The first character of the string value of <b>RS</b> shall be the input record separator; a <newline> by default. If <b>RS</b> contains more than one character, the results are unspecified. If <b>RS</b> is null, then records are separated by sequences consisting of a <newline> plus one or more blank lines, leading or trailing blank lines shall not result in empty records at the beginning or end of the input, and a <newline> shall always be a field separator, no matter what the value of <b>FS</b> is. |
| 80115 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80116 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80117 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80118 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80119 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80120 | <b>RSTART</b>  | The starting position of the string matched by the <b>match</b> function, numbering from 1. This shall always be equivalent to the return value of the <b>match</b> function.                                                                                                                                                                                                                                                                                                                                            |
| 80121 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 80122 | <b>SUBSEP</b>  | The subscript separator string for multi-dimensional arrays; the default value is implementation-defined.                                                                                                                                                                                                                                                                                                                                                                                                                |
| 80123 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

#### 80124 **Regular Expressions**

80125 The *awk* utility shall make use of the extended regular expression notation (see XBD [Section 9.4](#),  
80126 on page 188) except that it shall allow the use of C-language conventions for escaping special  
80127 characters within the EREs, as specified in the table in XBD [Chapter 5](#) (on page 121) ('\\',  
80128 '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v') and the following table; these escape sequences  
80129 shall be recognized both inside and outside bracket expressions. Note that records need not be  
80130 separated by <newline> characters and string constants can contain <newline> characters, so  
80131 even the "\\n" sequence is valid in *awk* EREs. Using a <slash> character within an ERE requires  
80132 the escaping shown in the following table.

80133

Table 4-2 Escape Sequences in *awk*

| Escape Sequence | Description                                                                                                                                                                                                             | Meaning                                                                                                                                                                                                                         |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \"              | <backslash> <quotation-mark>                                                                                                                                                                                            | <quotation-mark> character                                                                                                                                                                                                      |
| \/              | <backslash> <slash>                                                                                                                                                                                                     | <slash> character                                                                                                                                                                                                               |
| \ddd            | A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined. | The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte. |
| \c              | A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 121) ('\ ', '\a', '\b', '\f', '\n', '\r', '\t', '\v').                                          | Undefined                                                                                                                                                                                                                       |

80150 A regular expression can be matched against a specific field or string by using one of the two  
80151 regular expression matching operators, '~' and '!~'. These operators shall interpret their  
80152 right-hand operand as a regular expression and their left-hand operand as a string. If the regular  
80153 expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~'  
80154 expression shall evaluate to a value of 0. (The regular expression matching operation is as  
80155 defined by the term *matched* in XBD Section 9.1 (on page 181), where a match occurs on any part  
80156 of the string unless the regular expression is limited with the <circumflex> or <dollar-sign>  
80157 special characters.) If the regular expression does not match the string, the '~' expression shall  
80158 evaluate to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand  
80159 operand is any expression other than the lexical token **ERE**, the string value of the expression  
80160 shall be interpreted as an extended regular expression, including the escape conventions  
80161 described above. Note that these same escape conventions shall also be applied in determining  
80162 the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time  
80163 when a string literal is used in this context.

80164 When an **ERE** token appears as an expression in any context other than as the right-hand of the  
80165 '~' or '!~' operator or as one of the built-in function arguments described below, the value of  
80166 the resulting expression shall be the equivalent of:

```
80167 $0 ~ /ere/
```

80168 The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function  
80169 (see **String Functions**, on page 2497) shall be interpreted as extended regular expressions. These  
80170 can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner  
80171 as the right-hand side of the '~' or '!~' operator.

80172 An extended regular expression can be used to separate fields by assigning a string containing  
80173 the expression to the built-in variable **FS**, either directly or as a consequence of using the **-F**  
80174 *sepstring* option. The default value of the **FS** variable shall be a single <space>. The following  
80175 describes **FS** behavior:

- 80176 1. If **FS** is a null string, the behavior is unspecified.

- 80177 2. If **FS** is a single character:
- 80178 a. If **FS** is <space>, skip leading and trailing <blank> and <newline> characters;
- 80179 fields shall be delimited by sets of one or more <blank> or <newline> characters.
- 80180 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single
- 80181 occurrence of *c*.
- 80182 3. Otherwise, the string value of **FS** shall be considered to be an extended regular
- 80183 expression. Each occurrence of a sequence matching the extended regular expression shall
- 80184 delimit fields.

80185 Except for the '~' and '!~' operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,

80186 ERE matching shall be based on input records; that is, record separator characters (the first

80187 character of the value of the variable **RS**, <newline> by default) cannot be embedded in the

80188 expression, and no expression shall match the record separator character. If the record separator

80189 is not <newline>, <newline> characters embedded in the expression can be matched. For the

80190 '~' and '!~' operators, and in those four built-in functions, ERE matching shall be based on

80191 text strings; that is, any character (including <newline> and the record separator) can be

80192 embedded in the pattern, and an appropriate pattern shall match any character. However, in all

80193 *awk* ERE matching, the use of one or more NUL characters in the pattern, input record, or text

80194 string produces undefined results.

## 80195 **Patterns**

80196 A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or

80197 one of the two special patterns **BEGIN** or **END**.

## 80198 **Special Patterns**

80199 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern

80200 shall be matched once and its associated action executed before the first record of input is read ‡

80201 except possibly by use of the **getline** function (see [Input/Output and General Functions](#), on

80202 page 2498) in a prior **BEGIN** action ‡and before command line assignment is done. Each **END**

80203 pattern shall be matched once and its associated action executed after the last record of input has

80204 been read. These two patterns shall have associated actions.

80205 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns

80206 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order

80207 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern

80208 in a program.

80209 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action

80210 contains no **getline** function, *awk* shall exit without reading its input when the last statement in

80211 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern

80212 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the

80213 statements in the **END** actions are executed.

80214 **Expression Patterns**

80215 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the  
 80216 result is true, the pattern shall be considered to match, and the associated action (if any) shall be  
 80217 executed. If the result is false, the action shall not be executed.

80218 **Pattern Ranges**

80219 A pattern range consists of two expressions separated by a comma; in this case, the action shall  
 80220 be performed for all records between a match of the first expression and the following match of  
 80221 the second expression, inclusive. At this point, the pattern range can be repeated starting at  
 80222 input records subsequent to the end of the matched range.

80223 **Actions**

80224 An action is a sequence of statements as shown in the grammar in [Grammar](#) (on page 2500).  
 80225 Any single statement can be replaced by a statement list enclosed in curly braces. The  
 80226 application shall ensure that statements in a statement list are separated by <newline> or  
 80227 <semicolon> characters. Statements in a statement list shall be executed sequentially in the order  
 80228 that they appear.

80229 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero  
 80230 or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement  
 80231 following the **else** shall be executed.

80232 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard  
 80233 (see [Section 1.1.2](#), on page 2331), except that the Boolean expressions shall be treated as  
 80234 described in [Expressions in awk](#) (on page 2485), and except in the case of:

```
80235 for (variable in array)
```

80236 which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of  
 80237 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**  
 80238 statement occurs outside of a loop, the behavior is undefined.

80239 The **delete** statement shall remove an individual array element. Thus, the following code deletes  
 80240 an entire array:

```
80241 for (index in array)
80242 delete array[index]
```

80243 The **next** statement shall cause all further processing of the current input record to be  
 80244 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or  
 80245 **END** action.

80246 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program  
 80247 source and then terminate the program without reading further input. An **exit** statement inside  
 80248 an **END** action shall terminate the program without further execution of **END** actions. If an  
 80249 expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*,  
 80250 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is  
 80251 executed.

80252 **Output Statements**

80253 Both **print** and **printf** statements shall write to standard output by default. The output shall be  
 80254 written to the location specified by *output\_redirection* if one is supplied, as follows:

```
80255 > expression
80256 >> expression
80257 | expression
```

80258 In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into  
 80259 which to write (for '>' or ">>") or as a command to be executed (for '|'). Using the first two  
 80260 forms, if the file of that name is not currently open, it shall be opened, creating it if necessary  
 80261 and using the first form, truncating the file. The output then shall be appended to the file. As  
 80262 long as the file remains open, subsequent calls in which *expression* evaluates to the same string  
 80263 value shall simply append output to the file. The file remains open until the **close** function (see  
 80264 [Input/Output and General Functions](#), on page 2498) is called with an expression that evaluates  
 80265 to the same string value.

80266 The third form shall write output onto a stream piped to the input of a command. The stream  
 80267 shall be created if no stream is currently open with the value of *expression* as its command name.  
 80268 The stream created shall be equivalent to one created by a call to the *popen()* function defined in  
 80269 the System Interfaces volume of POSIX.1-2017 with the value of *expression* as the *command*  
 80270 argument and a value of *w* as the *mode* argument. As long as the stream remains open,  
 80271 subsequent calls in which *expression* evaluates to the same string value shall write output to the  
 80272 existing stream. The stream shall remain open until the **close** function (see [Input/Output and  
 80273 General Functions](#), on page 2498) is called with an expression that evaluates to the same string  
 80274 value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in  
 80275 the System Interfaces volume of POSIX.1-2017.

80276 As described in detail by the grammar in [Grammar](#) (on page 2500), these output statements shall  
 80277 take a <comma>-separated list of *expressions* referred to in the grammar by the non-terminal  
 80278 symbols **expr\_list**, **print\_expr\_list**, or **print\_expr\_list\_opt**. This list is referred to here as the  
 80279 *expression list*, and each member is referred to as an *expression argument*.

80280 The **print** statement shall write the value of each expression argument onto the indicated output  
 80281 stream separated by the current output field separator (see variable **OFS** above), and terminated  
 80282 by the output record separator (see variable **ORS** above). All expression arguments shall be  
 80283 taken as strings, being converted if necessary; this conversion shall be as described in  
 80284 [Expressions in awk](#) (on page 2485), with the exception that the **printf** format in **OFMT** shall be  
 80285 used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole  
 80286 input record (\$0).

80287 The **printf** statement shall produce output based on a notation similar to the File Format  
 80288 Notation used to describe file formats in this volume of POSIX.1-2017 (see XBD [Chapter 5](#), on  
 80289 page 121). Output shall be produced as specified with the first *expression* argument as the string  
 80290 *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the  
 80291 following exceptions:

- 80292 1. The *format* shall be an actual character string rather than a graphical representation.  
 80293 Therefore, it cannot contain empty character positions. The <space> in the *format* string,  
 80294 in any context other than a *flag* of a conversion specification, shall be treated as an  
 80295 ordinary character that is copied to the output.
- 80296 2. If the character set contains a ' $\Delta$ ' character and that character appears in the *format*  
 80297 string, it shall be treated as an ordinary character that is copied to the output.

- 80298 3. The *escape sequences* beginning with a <backslash> character shall be treated as sequences  
 80299 of ordinary characters that are copied to the output. Note that these same sequences shall  
 80300 be interpreted lexically by *awk* when they appear in literal strings, but they shall not be  
 80301 treated specially by the **printf** statement.
- 80302 4. A *field width* or *precision* can be specified as the '\*' character instead of a digit string. In  
 80303 this case the next argument from the expression list shall be fetched and its numeric value  
 80304 taken as the field width or precision.
- 80305 5. The implementation shall not precede or follow output from the *d* or *u* conversion  
 80306 specifier characters with <blank> characters not specified by the *format* string.
- 80307 6. The implementation shall not precede output from the *o* conversion specifier character  
 80308 with leading zeros not specified by the *format* string.
- 80309 7. For the *c* conversion specifier character: if the argument has a numeric value, the  
 80310 character whose encoding is that value shall be output. If the value is zero or is not the  
 80311 encoding of any character in the character set, the behavior is undefined. If the argument  
 80312 does not have a numeric value, the first character of the string value shall be output; if the  
 80313 string does not contain any characters, the behavior is undefined.
- 80314 8. For each conversion specification that consumes an argument, the next expression  
 80315 argument shall be evaluated. With the exception of the *c* conversion specifier character,  
 80316 the value shall be converted (according to the rules specified in [Expressions in awk](#), on  
 80317 page 2485) to the appropriate type for the conversion specification.
- 80318 9. If there are insufficient expression arguments to satisfy all the conversion specifications in  
 80319 the *format* string, the behavior is undefined.
- 80320 10. If any character sequence in the *format* string begins with a '%' character, but does not  
 80321 form a valid conversion specification, the behavior is unspecified.

80322 Both **print** and **printf** can output at least {LINE\_MAX} bytes.

## 80323 Functions

80324 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and  
 80325 general.

## 80326 Arithmetic Functions

80327 The arithmetic functions, except for **int**, shall be based on the ISO C standard (see [Section 1.1.2](#),  
 80328 on page 2331). The behavior is undefined in cases where the ISO C standard specifies that an  
 80329 error be returned or that the behavior is undefined. Although the grammar (see [Grammar](#), on  
 80330 page 2500) permits built-in functions to appear with no arguments or parentheses, unless the  
 80331 argument or parentheses are indicated as optional in the following list (by displaying them  
 80332 within the "[ ]" brackets), such use is undefined.

80333 **atan2**(*y*,*x*) Return arctangent of *y*/*x* in radians in the range  $[-\pi, \pi]$ .

80334 **cos**(*x*) Return cosine of *x*, where *x* is in radians.

80335 **sin**(*x*) Return sine of *x*, where *x* is in radians.

80336 **exp**(*x*) Return the exponential function of *x*.

80337 **log**(*x*) Return the natural logarithm of *x*.

- 80338        **sqrt**(*x*)        Return the square root of *x*.
- 80339        **int**(*x*)        Return the argument truncated to an integer. Truncation shall be toward 0 when  
80340                    *x*>0.
- 80341        **rand**()        Return a random number *n*, such that  $0 \leq n < 1$ .
- 80342        **srand**([*expr*]) Set the seed value for *rand* to *expr* or use the time of day if *expr* is omitted. The  
80343                    previous seed value shall be returned.

### 80344        **String Functions**

80345        The string functions in the following list shall be supported. Although the grammar (see  
80346        [Grammar](#), on page 2500) permits built-in functions to appear with no arguments or parentheses,  
80347        unless the argument or parentheses are indicated as optional in the following list (by displaying  
80348        them within the "[ ]" brackets), such use is undefined.

- 80349        **gsub**(*ere, repl*[, *in*])  
80350                    Behave like **sub** (see below), except that it shall replace all occurrences of the  
80351                    regular expression (like the *ed* utility global substitute) in \$0 or in the *in* argument,  
80352                    when specified.
- 80353        **index**(*s, t*)        Return the position, in characters, numbering from 1, in string *s* where string *t* first  
80354                    occurs, or zero if it does not occur at all.
- 80355        **length**[(*l*)]        Return the length, in characters, of its argument taken as a string, or of the whole  
80356                    record, \$0, if there is no argument.
- 80357        **match**(*s, ere*)        Return the position, in characters, numbering from 1, in string *s* where the  
80358                    extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART  
80359                    shall be set to the starting position (which is the same as the returned value), zero  
80360                    if no match is found; RLENGTH shall be set to the length of the matched string, -1  
80361                    if no match is found.
- 80362        **split**(*s, a*[, *fs* ])  
80363                    Split the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and return *n*. All elements  
80364                    of the array shall be deleted before the split is performed. The separation shall be  
80365                    done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array  
80366                    element shall have a string value when created and, if appropriate, the array  
80367                    element shall be considered a numeric string (see [Expressions in awk](#), on page  
80368                    2485). The effect of a null string as the value of *fs* is unspecified.
- 80369        **sprintf**(*fmt, expr, expr, ...*)  
80370                    Format the expressions according to the **printf** format given by *fmt* and return the  
80371                    resulting string.
- 80372        **sub**(*ere, repl*[, *in* ])  
80373                    Substitute the string *repl* in place of the first instance of the extended regular  
80374                    expression *ERE* in string *in* and return the number of substitutions. An  
80375                    <ampersand> ('&') appearing in the string *repl* shall be replaced by the string  
80376                    from *in* that matches the ERE. An <ampersand> preceded with a <backslash> shall  
80377                    be interpreted as the literal <ampersand> character. An occurrence of two  
80378                    consecutive <backslash> characters shall be interpreted as just a single literal  
80379                    <backslash> character. Any other occurrence of a <backslash> (for example,  
80380                    preceding any other character) shall be treated as a literal <backslash> character.  
80381                    Note that if *repl* is a string literal (the lexical token **STRING**; see [Grammar](#), on page  
80382                    2500), the handling of the <ampersand> character occurs after any lexical



80383 processing, including any lexical <backslash>-escape sequence processing. If *in* is  
 80384 specified and it is not an lvalue (see [Expressions in awk](#), on page 2485), the  
 80385 behavior is undefined. If *in* is omitted, *awk* shall use the current record (\$0) in its  
 80386 place.

80387 **substr**(*s*, *m*[, *n* ])

80388 Return the at most *n*-character substring of *s* that begins at position *m*, numbering  
 80389 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,  
 80390 the length of the substring shall be limited by the length of the string *s*.

80391 **tolower**(*s*) Return a string based on the string *s*. Each character in *s* that is an uppercase letter  
 80392 specified to have a **tolower** mapping by the *LC\_CTYPE* category of the current  
 80393 locale shall be replaced in the returned string by the lowercase letter specified by  
 80394 the mapping. Other characters in *s* shall be unchanged in the returned string.

80395 **toupper**(*s*) Return a string based on the string *s*. Each character in *s* that is a lowercase letter  
 80396 specified to have a **toupper** mapping by the *LC\_CTYPE* category of the current  
 80397 locale is replaced in the returned string by the uppercase letter specified by the  
 80398 mapping. Other characters in *s* are unchanged in the returned string.

80399 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued  
 80400 expression that is a regular expression as defined in [Regular Expressions](#) (on page 2491).

## 80401 Input/Output and General Functions

80402 The input/output and general functions are:

80403 **close**(*expression*)

80404 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with  
 80405 the same string-valued *expression*. The limit on the number of open *expression*  
 80406 arguments is implementation-defined. If the close was successful, the function  
 80407 shall return zero; otherwise, it shall return non-zero.

80408 *expression* | **getline** [*var*]

80409 Read a record of input from a stream piped from the output of a command. The  
 80410 stream shall be created if no stream is currently open with the value of *expression* as  
 80411 its command name. The stream created shall be equivalent to one created by a call  
 80412 to the *popen*() function with the value of *expression* as the *command* argument and a  
 80413 value of *r* as the *mode* argument. As long as the stream remains open, subsequent  
 80414 calls in which *expression* evaluates to the same string value shall read subsequent  
 80415 records from the stream. The stream shall remain open until the **close** function is  
 80416 called with an expression that evaluates to the same string value. At that time, the  
 80417 stream shall be closed as if by a call to the *pclose*() function. If *var* is omitted, \$0 and  
 80418 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 80419 a numeric string (see [Expressions in awk](#), on page 2485).

80420 The **getline** operator can form ambiguous constructs when there are  
 80421 unparenthesized operators (including concatenate) to the left of the '|' (to the  
 80422 beginning of the expression containing **getline**). In the context of the '\$' operator,  
 80423 '|' shall behave as if it had a lower precedence than '\$'. The result of evaluating  
 80424 other operators is unspecified, and conforming applications shall parenthesize  
 80425 properly all such usages.

80426 **getline** Set \$0 to the next input record from the current input file. This form of **getline** shall  
 80427 set the **NF**, **NR**, and **FNR** variables.

80428 **getline** *var* Set variable *var* to the next input record from the current input file and, if  
 80429 appropriate, *var* shall be considered a numeric string (see [Expressions in awk](#), on  
 80430 page 2485). This form of **getline** shall set the **FNR** and **NR** variables.

80431 **getline** [*var*] < *expression*  
 80432 Read the next record of input from a named file. The *expression* shall be evaluated  
 80433 to produce a string that is used as a pathname. If the file of that name is not  
 80434 currently open, it shall be opened. As long as the stream remains open, subsequent  
 80435 calls in which *expression* evaluates to the same string value shall read subsequent  
 80436 records from the file. The file shall remain open until the **close** function is called  
 80437 with an expression that evaluates to the same string value. If *var* is omitted, **\$0** and  
 80438 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered  
 80439 a numeric string (see [Expressions in awk](#), on page 2485).

80440 The **getline** operator can form ambiguous constructs when there are  
 80441 unparenthesized binary operators (including concatenate) to the right of the '<'  
 80442 (up to the end of the expression containing the **getline**). The result of evaluating  
 80443 such a construct is unspecified, and conforming applications shall parenthesize  
 80444 properly all such usages.

80445 **system**(*expression*)  
 80446 Execute the command given by *expression* in a manner equivalent to the *system()*  
 80447 function defined in the System Interfaces volume of POSIX.1-2017 and return the  
 80448 exit status of the command.

80449 All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

80450 Where strings are used as the name of a file or pipeline, the application shall ensure that the  
 80451 strings are textually identical. The terminology “same string value” implies that “equivalent  
 80452 strings”, even those that differ only by <space> characters, represent different files.

### 80453 User-Defined Functions

80454 The *awk* language also provides user-defined functions. Such functions can be defined as:

```
80455 function name([parameter, ...]) { statements }
```

80456 A function can be referred to anywhere in an *awk* program; in particular, its use can precede its  
 80457 definition. The scope of a function is global.

80458 Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an  
 80459 array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is  
 80460 passed as a parameter that the function uses as an array. Function parameters shall be passed by  
 80461 value if scalar and by reference if array name.

80462 The number of parameters in the function definition need not match the number of parameters  
 80463 in the function call. Excess formal parameters can be used as local variables. If fewer arguments  
 80464 are supplied in a function call than are in the function definition, the extra parameters that are  
 80465 used in the function body as scalars shall evaluate to the uninitialized value until they are  
 80466 otherwise initialized, and the extra parameters that are used in the function body as arrays shall  
 80467 be treated as uninitialized arrays where each element evaluates to the uninitialized value until  
 80468 otherwise initialized.

80469 When invoking a function, no white space can be placed between the function name and the  
 80470 opening parenthesis. Function calls can be nested and recursive calls can be made upon  
 80471 functions. Upon return from any nested or recursive function call, the values of all of the calling  
 80472 function's parameters shall be unchanged, except for array parameters passed by reference. The

80473 **return** statement can be used to return a value. If a **return** statement appears outside of a  
80474 function definition, the behavior is undefined.

80475 In the function definition, <newline> characters shall be optional before the opening brace and  
80476 after the closing brace. Function definitions can appear anywhere in the program where a  
80477 *pattern-action* pair is allowed.

## 80478 Grammar

80479 The grammar in this section and the lexical conventions in the following section shall together  
80480 describe the syntax for *awk* programs. The general conventions for this style of grammar are  
80481 described in [Section 1.3](#) (on page 2335). A valid program can be represented as the non-terminal  
80482 symbol *program* in the grammar. This formal syntax shall take precedence over the preceding  
80483 text syntax description.

```
80484 %token NAME NUMBER STRING ERE
80485 %token FUNC_NAME /* Name followed by '(' without white space. */

80486 /* Keywords */
80487 %token Begin End
80488 /* 'BEGIN' 'END' */

80489 %token Break Continue Delete Do Else
80490 /* 'break' 'continue' 'delete' 'do' 'else' */

80491 %token Exit For Function If In
80492 /* 'exit' 'for' 'function' 'if' 'in' */

80493 %token Next Print Printf Return While
80494 /* 'next' 'print' 'printf' 'return' 'while' */

80495 /* Reserved function names */
80496 %token BUILTIN_FUNC_NAME
80497 /* One token for the following:
80498 * atan2 cos sin exp log sqrt int rand srand
80499 * gsub index length match split sprintf sub
80500 * substr tolower toupper close system
80501 */
80502 %token GETLINE
80503 /* Syntactically different from other built-ins. */

80504 /* Two-character tokens. */
80505 %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
80506 /* '+=' '-=' '*=' '/=' '%=' '^=' */

80507 %token OR AND NO_MATCH EQ LE GE NE INCR DECR APPEND
80508 /* '||' '&&' '!~' '==' '<=' '>=' '!=' '++' '--' '>>' */

80509 /* One-character tokens. */
80510 %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
80511 %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

80512 %start program
80513 %%

80514 program : item_list
80515 | item_list item
80516 ;
```

```

80517 item_list : /* empty */
80518 | item_list item terminator
80519 ;
80520 item : action
80521 | pattern action
80522 | normal_pattern
80523 | Function NAME '(' param_list_opt ')'
80524 newline_opt action
80525 | Function FUNC_NAME '(' param_list_opt ')'
80526 newline_opt action
80527 ;
80528 param_list_opt : /* empty */
80529 | param_list
80530 ;
80531 param_list : NAME
80532 | param_list ',' NAME
80533 ;
80534 pattern : normal_pattern
80535 | special_pattern
80536 ;
80537 normal_pattern : expr
80538 | expr ',' newline_opt expr
80539 ;
80540 special_pattern : Begin
80541 | End
80542 ;
80543 action : '{' newline_opt
80544 | '{' newline_opt terminated_statement_list
80545 | '{' newline_opt unterminated_statement_list
80546 ;
80547 terminator : terminator NEWLINE
80548 | ';'
80549 | NEWLINE
80550 ;
80551 terminated_statement_list : terminated_statement
80552 | terminated_statement_list terminated_statement
80553 ;
80554 unterminated_statement_list : unterminated_statement
80555 | terminated_statement_list unterminated_statement
80556 ;
80557 terminated_statement : action newline_opt
80558 | If '(' expr ')' newline_opt terminated_statement
80559 | If '(' expr ')' newline_opt terminated_statement
80560 Else newline_opt terminated_statement
80561 | While '(' expr ')' newline_opt terminated_statement
80562 | For '(' simple_statement_opt ';'

```

```

80563 expr_opt ';' simple_statement_opt ')' newline_opt
80564 terminated_statement
80565 | For '(' NAME In NAME ')' newline_opt
80566 terminated_statement
80567 | ';' newline_opt
80568 | terminatable_statement NEWLINE newline_opt
80569 | terminatable_statement ';' newline_opt
80570 ;

80571 unterminated_statement : terminatable_statement
80572 | If '(' expr ')' newline_opt unterminated_statement
80573 | If '(' expr ')' newline_opt terminated_statement
80574 | Else newline_opt unterminated_statement
80575 | While '(' expr ')' newline_opt unterminated_statement
80576 | For '(' simple_statement_opt ';'
80577 | expr_opt ';' simple_statement_opt ')' newline_opt
80578 | unterminated_statement
80579 | For '(' NAME In NAME ')' newline_opt
80580 | unterminated_statement
80581 ;

80582 terminatable_statement : simple_statement
80583 | Break
80584 | Continue
80585 | Next
80586 | Exit expr_opt
80587 | Return expr_opt
80588 | Do newline_opt terminated_statement While '(' expr ')'
80589 ;

80590 simple_statement_opt : /* empty */
80591 | simple_statement
80592 ;

80593 simple_statement : Delete NAME '[' expr_list ']'
80594 | expr
80595 | print_statement
80596 ;

80597 print_statement : simple_print_statement
80598 | simple_print_statement output_redirection
80599 ;

80600 simple_print_statement : Print print_expr_list_opt
80601 | Print '(' multiple_expr_list ')'
80602 | Printf print_expr_list
80603 | Printf '(' multiple_expr_list ')'
80604 ;

80605 output_redirection : '>' expr
80606 | APPEND expr
80607 | '|' expr
80608 ;

80609 expr_list_opt : /* empty */
80610 | expr_list

```

```

80611 ;
80612 expr_list : expr
80613 | multiple_expr_list
80614 ;
80615 multiple_expr_list : expr ',' newline_opt expr
80616 | multiple_expr_list ',' newline_opt expr
80617 ;
80618 expr_opt : /* empty */
80619 | expr
80620 ;
80621 expr : unary_expr
80622 | non_unary_expr
80623 ;
80624 unary_expr : '+' expr
80625 | '-' expr
80626 | unary_expr '^' expr
80627 | unary_expr '*' expr
80628 | unary_expr '/' expr
80629 | unary_expr '%' expr
80630 | unary_expr '+' expr
80631 | unary_expr '-' expr
80632 | unary_expr non_unary_expr
80633 | unary_expr '<' expr
80634 | unary_expr LE expr
80635 | unary_expr NE expr
80636 | unary_expr EQ expr
80637 | unary_expr '>' expr
80638 | unary_expr GE expr
80639 | unary_expr '~' expr
80640 | unary_expr NO_MATCH expr
80641 | unary_expr In NAME
80642 | unary_expr AND newline_opt expr
80643 | unary_expr OR newline_opt expr
80644 | unary_expr '?' expr ':' expr
80645 | unary_input_function
80646 ;
80647 non_unary_expr : '(' expr ')'
80648 | '!' expr
80649 | non_unary_expr '^' expr
80650 | non_unary_expr '*' expr
80651 | non_unary_expr '/' expr
80652 | non_unary_expr '%' expr
80653 | non_unary_expr '+' expr
80654 | non_unary_expr '-' expr
80655 | non_unary_expr non_unary_expr
80656 | non_unary_expr '<' expr
80657 | non_unary_expr LE expr
80658 | non_unary_expr NE expr
80659 | non_unary_expr EQ expr

```

```

80660 | non_unary_expr '>' expr
80661 | non_unary_expr GE expr
80662 | non_unary_expr '~' expr
80663 | non_unary_expr NO_MATCH expr
80664 | non_unary_expr In NAME
80665 | '(' multiple_expr_list ')' In NAME
80666 | non_unary_expr AND newline_opt expr
80667 | non_unary_expr OR newline_opt expr
80668 | non_unary_expr '?' expr ':' expr
80669 | NUMBER
80670 | STRING
80671 | lvalue
80672 | ERE
80673 | lvalue INCR
80674 | lvalue DECR
80675 | INCR lvalue
80676 | DECR lvalue
80677 | lvalue POW_ASSIGN expr
80678 | lvalue MOD_ASSIGN expr
80679 | lvalue MUL_ASSIGN expr
80680 | lvalue DIV_ASSIGN expr
80681 | lvalue ADD_ASSIGN expr
80682 | lvalue SUB_ASSIGN expr
80683 | lvalue '=' expr
80684 | FUNC_NAME '(' expr_list_opt ')'
80685 | /* no white space allowed before '(' */
80686 | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
80687 | BUILTIN_FUNC_NAME
80688 | non_unary_input_function
80689 ;

80690 print_expr_list_opt : /* empty */
80691 | print_expr_list
80692 ;

80693 print_expr_list : print_expr
80694 | print_expr_list ',' newline_opt print_expr
80695 ;

80696 print_expr : unary_print_expr
80697 | non_unary_print_expr
80698 ;

80699 unary_print_expr : '+' print_expr
80700 | '-' print_expr
80701 | unary_print_expr '^' print_expr
80702 | unary_print_expr '*' print_expr
80703 | unary_print_expr '/' print_expr
80704 | unary_print_expr '%' print_expr
80705 | unary_print_expr '+' print_expr
80706 | unary_print_expr '-' print_expr
80707 | unary_print_expr non_unary_print_expr
80708 | unary_print_expr '~' print_expr
80709 | unary_print_expr NO_MATCH print_expr

```

```

80710 | unary_print_expr In NAME
80711 | unary_print_expr AND newline_opt print_expr
80712 | unary_print_expr OR newline_opt print_expr
80713 | unary_print_expr '?' print_expr ':' print_expr
80714 ;

80715 non_unary_print_expr : '(' expr ')'
80716 | '!' print_expr
80717 | non_unary_print_expr '^' print_expr
80718 | non_unary_print_expr '*' print_expr
80719 | non_unary_print_expr '/' print_expr
80720 | non_unary_print_expr '%' print_expr
80721 | non_unary_print_expr '+' print_expr
80722 | non_unary_print_expr '-' print_expr
80723 | non_unary_print_expr non_unary_print_expr
80724 | non_unary_print_expr '~' print_expr
80725 | non_unary_print_expr NO_MATCH print_expr
80726 | non_unary_print_expr In NAME
80727 | '(' multiple_expr_list ')' In NAME
80728 | non_unary_print_expr AND newline_opt print_expr
80729 | non_unary_print_expr OR newline_opt print_expr
80730 | non_unary_print_expr '?' print_expr ':' print_expr
80731 | NUMBER
80732 | STRING
80733 | lvalue
80734 | ERE
80735 | lvalue INCR
80736 | lvalue DECR
80737 | INCR lvalue
80738 | DECR lvalue
80739 | lvalue POW_ASSIGN print_expr
80740 | lvalue MOD_ASSIGN print_expr
80741 | lvalue MUL_ASSIGN print_expr
80742 | lvalue DIV_ASSIGN print_expr
80743 | lvalue ADD_ASSIGN print_expr
80744 | lvalue SUB_ASSIGN print_expr
80745 | lvalue '=' print_expr
80746 | FUNC_NAME '(' expr_list_opt ')'
80747 /* no white space allowed before '(' */
80748 | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
80749 | BUILTIN_FUNC_NAME
80750 ;

80751 lvalue : NAME
80752 | NAME '[' expr_list ']'
80753 | '$' expr
80754 ;

80755 non_unary_input_function : simple_get
80756 | simple_get '<' expr
80757 | non_unary_expr '|' simple_get
80758 ;

80759 unary_input_function : unary_expr '|' simple_get

```



```

80760 ;
80761 simple_get : GETLINE
80762 | GETLINE lvalue
80763 ;
80764 newline_opt : /* empty */
80765 | newline_opt NEWLINE
80766 ;

```

80767 This grammar has several ambiguities that shall be resolved as follows:

80768 Operator precedence and associativity shall be as described in [Table 4-1](#) (on page 2485).

80769 In case of ambiguity, an **else** shall be associated with the most immediately preceding **if**  
80770 that would satisfy the grammar.

80771 In some contexts, a `<slash>` ('/') that is used to surround an ERE could also be the  
80772 division operator. This shall be resolved in such a way that wherever the division operator  
80773 could appear, a `<slash>` is assumed to be the division operator. (There is no unary division  
80774 operator.)

80775 Each expression in an *awk* program shall conform to the precedence and associativity rules, even  
80776 when this is not needed to resolve an ambiguity. For example, because '\$' has higher  
80777 precedence than '++', the string "\$x++--" is not a valid *awk* expression, even though it is  
80778 unambiguously parsed by the grammar as "\$ (x++)--".

80779 One convention that might not be obvious from the formal grammar is where `<newline>`  
80780 characters are acceptable. There are several obvious placements such as terminating a statement,  
80781 and a `<backslash>` can be used to escape `<newline>` characters between any lexical tokens. In  
80782 addition, `<newline>` characters without `<backslash>` characters can follow a comma, an open  
80783 brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else**  
80784 keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```

80785 { print $1,
80786 $2 }

```

## 80787 Lexical Conventions

80788 The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as  
80789 follows:

- 80790 1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a  
80791 given point.
- 80792 2. A comment shall consist of any characters beginning with the `<number-sign>` character  
80793 and terminated by, but excluding the next occurrence of, a `<newline>`. Comments shall  
80794 have no effect, except to delimit lexical tokens.
- 80795 3. The `<newline>` shall be recognized as the token **NEWLINE**.
- 80796 4. A `<backslash>` character immediately followed by a `<newline>` shall have no effect.
- 80797 5. The token **STRING** shall represent a string constant. A string constant shall begin with  
80798 the character `'`. Within a string constant, a `<backslash>` character shall be considered  
80799 to begin an escape sequence as specified in the table in [XBD Chapter 5](#) (on page 121)  
80800 (`'\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). In addition, the escape sequences  
80801 in [Table 4-2](#) (on page 2492) shall be recognized. A `<newline>` shall not occur within a  
80802 string constant. A string constant shall be terminated by the first unescaped occurrence of

80803 the character `'` after the one that begins the string constant. The value of the string  
 80804 shall be the sequence of all unescaped characters and values of escape sequences  
 80805 between, but not including, the two delimiting `'` characters.

80806 6. The token **ERE** represents an extended regular expression constant. An ERE constant  
 80807 shall begin with the `<slash>` character. Within an ERE constant, a `<backslash>` character  
 80808 shall be considered to begin an escape sequence as specified in the table in XBD [Chapter 5](#)  
 80809 (on page 121). In addition, the escape sequences in [Table 4-2](#) (on page 2492) shall be  
 80810 recognized. The application shall ensure that a `<newline>` does not occur within an ERE  
 80811 constant. An ERE constant shall be terminated by the first unescaped occurrence of the  
 80812 `<slash>` character after the one that begins the ERE constant. The extended regular  
 80813 expression represented by the ERE constant shall be the sequence of all unescaped  
 80814 characters and values of escape sequences between, but not including, the two delimiting  
 80815 `<slash>` characters.

80816 7. A `<blank>` shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE**  
 80817 tokens.

80818 8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall  
 80819 either be equivalent to the **decimal-floating-constant** token as specified by the ISO C  
 80820 standard, or it shall be a sequence of decimal digits and shall be evaluated as an integer  
 80821 constant in decimal. In addition, implementations may accept numeric constants with the  
 80822 form and numeric value equivalent to the **hexadecimal-constant** and **hexadecimal-**  
 80823 **floating-constant** tokens as specified by the ISO C standard.

80824 If the value is too large or too small to be representable (see [Section 1.1.2](#), on page 2331),  
 80825 the behavior is undefined.

80826 9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see  
 80827 XBD [Section 6.1](#), on page 125), beginning with an `<underscore>` or alphabetic character,  
 80828 shall be considered a word.

80829 10. The following words are keywords that shall be recognized as individual tokens; the  
 80830 name of the token is the same as the keyword:

|       |                 |               |             |                 |              |               |
|-------|-----------------|---------------|-------------|-----------------|--------------|---------------|
| 80831 | <b>BEGIN</b>    | <b>delete</b> | <b>END</b>  | <b>function</b> | <b>in</b>    | <b>printf</b> |
| 80832 | <b>break</b>    | <b>do</b>     | <b>exit</b> | <b>getline</b>  | <b>next</b>  | <b>return</b> |
| 80833 | <b>continue</b> | <b>else</b>   | <b>for</b>  | <b>if</b>       | <b>print</b> | <b>while</b>  |

80834 11. The following words are names of built-in functions and shall be recognized as the token  
 80835 **BUILTIN\_FUNC\_NAME**:

|       |              |               |              |                |                |                |
|-------|--------------|---------------|--------------|----------------|----------------|----------------|
| 80836 | <b>atan2</b> | <b>gsub</b>   | <b>log</b>   | <b>split</b>   | <b>sub</b>     | <b>toupper</b> |
| 80837 | <b>close</b> | <b>index</b>  | <b>match</b> | <b>sprintf</b> | <b>substr</b>  |                |
| 80838 | <b>cos</b>   | <b>int</b>    | <b>rand</b>  | <b>sqrt</b>    | <b>system</b>  |                |
| 80839 | <b>exp</b>   | <b>length</b> | <b>sin</b>   | <b>srand</b>   | <b>tolower</b> |                |

80840 The above-listed keywords and names of built-in functions are considered reserved  
 80841 words.

80842 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in  
 80843 function and is not followed immediately (without any delimiters) by the `'` character.

80844 13. The token **FUNC\_NAME** shall consist of a word that is not a keyword or a name of a  
 80845 built-in function, followed immediately (without any delimiters) by the `'` character.  
 80846 The `'` character shall not be included as part of the token.

80847 14. The following two-character sequences shall be recognized as the named tokens:

| Token Name | Sequence | Token Name | Sequence |
|------------|----------|------------|----------|
| ADD_ASSIGN | +=       | NO_MATCH   | !~       |
| SUB_ASSIGN | --       | EQ         | ==       |
| MUL_ASSIGN | *=       | LE         | <=       |
| DIV_ASSIGN | /=       | GE         | >=       |
| MOD_ASSIGN | %=       | NE         | !=       |
| POW_ASSIGN | ^=       | INCR       | ++       |
| OR         |          | DECR       | --       |
| AND        | &&       | APPEND     | >>       |

80857 15. The following single characters shall be recognized as tokens whose names are the  
80858 character:

80859 <newline> { } ( ) [ ] , ; + - \* % ^ ! > < | ? : ~ \$ =

80860 There is a lexical ambiguity between the token **ERE** and the tokens **'/'** and **DIV\_ASSIGN**.  
80861 When an input sequence begins with a <slash> character in any syntactic context where the  
80862 token **'/'** or **DIV\_ASSIGN** could appear as the next token in a valid program, the longer of  
80863 those two tokens that can be recognized shall be recognized. In any other syntactic context  
80864 where the token **ERE** could appear as the next token in a valid program, the token **ERE** shall be  
80865 recognized.

#### 80866 EXIT STATUS

80867 The following exit values shall be returned:

80868 0 All input files were processed successfully.

80869 >0 An error occurred.

80870 The exit status can be altered within the program by using an **exit** expression.

#### 80871 CONSEQUENCES OF ERRORS

80872 If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a  
80873 diagnostic message to standard error and terminate without any further action.

80874 If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk*  
80875 program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

#### 80876 APPLICATION USAGE

80877 The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in  
80878 the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with  
80879 bytes.

80880 Because the concatenation operation is represented by adjacent expressions rather than an  
80881 explicit operator, it is often necessary to use parentheses to enforce the proper evaluation  
80882 precedence.

80883 When using *awk* to process pathnames, it is recommended that **LC\_ALL**, or at least **LC\_CTYPE**  
80884 and **LC\_COLLATE**, are set to **POSIX** or **C** in the environment, since pathnames can contain byte  
80885 sequences that do not form valid characters in some locales, in which case the utility's behavior  
80886 would be undefined. In the **POSIX** locale each byte is a valid single-byte character, and therefore  
80887 this problem is avoided.

80888 On implementations where the **"=="** operator checks if strings collate equally, applications  
80889 needing to check whether strings are identical can use:

80890 `length(a) == length(b) && index(a,b) == 1`

80891 On implementations where the "==" operator checks if strings are identical, applications  
80892 needing to check whether strings collate equally can use:

```
80893 a <= b && a >= b
```

#### 80894 EXAMPLES

80895 The *awk* program specified in the command line is most easily specified within single-quotes (for  
80896 example, *'program'*) for applications using *sh*, because *awk* programs commonly contain  
80897 characters that are special to the shell, including double-quotes. In the cases where an *awk*  
80898 program contains single-quote characters, it is usually easiest to specify most of the program as  
80899 strings within single-quotes concatenated by the shell with quoted single-quote characters. For  
80900 example:

```
80901 awk '/'\''/ { print "quote:", $0 }
```

80902 prints all lines from the standard input containing a single-quote character, prefixed with *quote*..

80903 The following are examples of simple *awk* programs:

- 80904 1. Write to the standard output all input lines for which field 3 is greater than 5:

```
80905 $3 > 5
```

- 80906 2. Write every tenth line:

```
80907 (NR % 10) == 0
```

- 80908 3. Write any line with a substring matching the regular expression:

```
80909 /(G|D)(2[0-9][[:alpha:]]*)/
```

- 80910 4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits  
80911 and characters. This example uses character classes **digit** and **alpha** to match language-  
80912 independent digit and alphabetic characters respectively:

```
80913 /(G|D)([[:digit:]][[:alpha:]]*)/
```

- 80914 5. Write any line in which the second field matches the regular expression and the fourth  
80915 field does not:

```
80916 $2 ~ /xyz/ && $4 !~ /xyz/
```

- 80917 6. Write any line in which the second field contains a <backslash>:

```
80918 $2 ~ /\
```

- 80919 7. Write any line in which the second field contains a <backslash>. Note that  
80920 <backslash>-escapes are interpreted twice; once in lexical processing of the string and  
80921 once in processing the regular expression:

```
80922 $2 ~ "\\
```

- 80923 8. Write the second to the last and the last field in each line. Separate the fields by a <colon>:

```
80924 {OFS=":";print $(NF-1), $NF}
```

- 80925 9. Write the line number and number of fields in each line. The three strings representing  
80926 the line number, the <colon>, and the number of fields are concatenated and that string is  
80927 written to standard output:

```
80928 {print NR ":" NF}
```

- 80929 10. Write lines longer than 72 characters:  
 80930 `length($0) > 72`
- 80931 11. Write the first two fields in opposite order separated by **OFS**:  
 80932 `{ print $2, $1 }`
- 80933 12. Same, with input fields separated by a <comma> or <space> and <tab> characters, or  
 80934 both:  
 80935 `BEGIN { FS = ",[ \t]*|[ \t]+" }`  
 80936 `{ print $2, $1 }`
- 80937 13. Add up the first column, print sum, and average:  
 80938 `{s += $1 }`  
 80939 `END {print "sum is ", s, " average is", s/NR}`
- 80940 14. Write fields in reverse order, one per line (many lines out for each line in):  
 80941 `{ for (i = NF; i > 0; --i) print $i }`
- 80942 15. Write all lines between occurrences of the strings **start** and **stop**:  
 80943 `/start/, /stop/`
- 80944 16. Write all lines whose first field is different from the previous one:  
 80945 `$1 != prev { print; prev = $1 }`
- 80946 17. Simulate *echo*:  
 80947 `BEGIN {`  
 80948 `for (i = 1; i < ARGV; ++i)`  
 80949 `printf("%s%s", ARGV[i], i==ARGC-1?"\n":" ")`  
 80950 `}`
- 80951 18. Write the path prefixes contained in the *PATH* environment variable, one per line:  
 80952 `BEGIN {`  
 80953 `n = split (ENVIRON["PATH"], path, ":")`  
 80954 `for (i = 1; i <= n; ++i)`  
 80955 `print path[i]`  
 80956 `}`
- 80957 19. If there is a file named **input** containing page headers of the form:  
 80958 **Page #**
- 80959 and a file named **program** that contains:  
 80960 `/Page/ { $2 = n++; }`  
 80961 `{ print }`
- 80962 then the command line:  
 80963 `awk -f program n=5 input`  
 80964 prints the file **input**, filling in page numbers starting at 5.

80965 **RATIONALE**

80966 This description is based on the new *awk*, “*nawk*”, (see the referenced *The AWK Programming*  
80967 *Language*), which introduced a number of new features to the historical *awk*:

- 80968 1. New keywords: **delete**, **do**, **function**, **return**
- 80969 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
- 80970 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
- 80971 4. New expression operators: **?**, **:**, **..**, **^**
- 80972 5. The **FS** variable and the third argument to **split**, now treated as extended regular  
80973 expressions.
- 80974 6. The operator precedence, changed to more closely match the C language. Two examples  
80975 of code that operate differently are:

```
80976 while (n /= 10 > 1) ...
80977 if (!"wk" ~ /bwk/) ...
```

80978 Several features have been added based on newer implementations of *awk*:

80979 Multiple instances of **-f** *profile* are permitted.

80980 The new option **-v** *assignment*.

80981 The new predefined variable **ENVIRON**.

80982 New built-in functions **toupper** and **tolower**.

80983 More formatting capabilities are added to **printf** to match the ISO C standard.

80984 Earlier versions of this standard required implementations to support multiple adjacent  
80985 <semicolon>s, lines with one or more <semicolon> before a rule (*pattern-action* pairs), and lines  
80986 with only <semicolon>(s). These are not required by this standard and are considered poor  
80987 programming practice, but can be accepted by an implementation of *awk* as an extension.

80988 The overall *awk* syntax has always been based on the C language, with a few features from the  
80989 shell command language and other sources. Because of this, it is not completely compatible with  
80990 any other language, which has caused confusion for some users. It is not the intent of the  
80991 standard developers to address such issues. A few relatively minor changes toward making the  
80992 language more compatible with the ISO C standard were made; most of these changes are based  
80993 on similar changes in recent implementations, as described above. There remain several C-  
80994 language conventions that are not in *awk*. One of the notable ones is the <comma> operator,  
80995 which is commonly used to specify multiple expressions in the C language **for** statement. Also,  
80996 there are various places where *awk* is more restrictive than the C language regarding the type of  
80997 expression that can be used in a given context. These limitations are due to the different features  
80998 that the *awk* language does provide.

80999 Regular expressions in *awk* have been extended somewhat from historical implementations to  
81000 make them a pure superset of extended regular expressions, as defined by POSIX.1-2017 (see  
81001 XBD [Section 9.4](#), on page 188). The main extensions are internationalization features and  
81002 interval expressions. Historical implementations of *awk* have long supported  
81003 <backslash>-escape sequences as an extension to extended regular expressions, and this  
81004 extension has been retained despite inconsistency with other utilities. The number of escape  
81005 sequences recognized in both extended regular expressions and strings has varied (generally  
81006 increasing with time) among implementations. The set specified by POSIX.1-2017 includes most  
81007 sequences known to be supported by popular implementations and by the ISO C standard. One  
81008 sequence that is not supported is hexadecimal value escapes beginning with '\x'. This would

81009 allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard.  
 81010 However, because this syntax has a non-deterministic length, it does not permit the subsequent  
 81011 character to be a hexadecimal digit. This limitation can be dealt with in the C language by the  
 81012 use of lexical string concatenation. In the *awk* language, concatenation could also be a solution  
 81013 for strings, but not for extended regular expressions (either lexical ERE tokens or strings used  
 81014 dynamically as regular expressions). Because of this limitation, the feature has not been added to  
 81015 POSIX.1-2017.

81016 When a string variable is used in a context where an extended regular expression normally  
 81017 appears (where the lexical token ERE is used in the grammar) the string does not contain the  
 81018 literal `<slash>` characters.

81019 Some versions of *awk* allow the form:

```
81020 func name(args, ...) { statements }
```

81021 This has been deprecated by the authors of the language, who asked that it not be specified.

81022 Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN**  
 81023 action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This  
 81024 behavior has not been documented, and it was not believed that it was necessary to standardize  
 81025 it.

81026 The specification of conversions between string and numeric values is much more detailed than  
 81027 in the documentation of historical implementations or in the referenced *The AWK Programming*  
 81028 *Language*. Although most of the behavior is designed to be intuitive, the details are necessary to  
 81029 ensure compatible behavior from different implementations. This is especially important in  
 81030 relational expressions since the types of the operands determine whether a string or numeric  
 81031 comparison is performed. From the perspective of an application developer, it is usually  
 81032 sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating  
 81033 a null string) when the type of an expression does not obviously match what is needed. The  
 81034 intent has been to specify historical practice in almost all cases. The one exception is that, in  
 81035 historical implementations, variables and constants maintain both string and numeric values  
 81036 after their original value is converted by any use. This means that referencing a variable or  
 81037 constant can have unexpected side-effects. For example, with historical implementations the  
 81038 following program:

```
81039 {
81040 a = "+2"
81041 b = 2
81042 if (NR % 2)
81043 c = a + b
81044 if (a == b)
81045 print "numeric comparison"
81046 else
81047 print "string comparison"
81048 }
```

81049 would perform a numeric comparison (and output numeric comparison) for each odd-  
 81050 numbered line, but perform a string comparison (and output string comparison) for each even-  
 81051 numbered line. POSIX.1-2017 ensures that comparisons will be numeric if necessary. With  
 81052 historical implementations, the following program:

```
81053 BEGIN {
81054 OFMT = "%e"
81055 print 3.14
```

```

81056 OFMT = "%f"
81057 print 3.14
81058 }

```

81059 would output "3.140000e+00" twice, because in the second **print** statement the constant  
 81060 "3.14" would have a string value from the previous conversion. POSIX.1-2017 requires that the  
 81061 output of the second **print** statement be "3.140000". The behavior of historical  
 81062 implementations was seen as too unintuitive and unpredictable.

81063 It was pointed out that with the rules contained in early drafts, the following script would print  
 81064 nothing:

```

81065 BEGIN {
81066 y[1.5] = 1
81067 OFMT = "%e"
81068 print y[1.5]
81069 }

```

81070 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to  
 81071 affecting output conversions of numbers to strings and **CONVFMT** is used for internal  
 81072 conversions, such as comparisons or array indexing. The default value is the same as that for  
 81073 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it  
 81074 will receive the historical behavior associated with internal string conversions.

81075 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other  
 81076 sources. Again the intent has been to specify historical practice. One convention that may not be  
 81077 obvious from the formal grammar as in other verbal descriptions is where <newline> characters  
 81078 are acceptable. There are several obvious placements such as terminating a statement, and a  
 81079 <backslash> can be used to escape <newline> characters between any lexical tokens. In addition,  
 81080 <newline> characters without <backslash> characters can follow a comma, an open brace, a  
 81081 logical AND operator ("&&"), a logical OR operator ("||"), the **do** keyword, the **else** keyword,  
 81082 and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```

81083 { print $1,
81084 $2 }

```

81085 The requirement that *awk* add a trailing <newline> to the program argument text is to simplify  
 81086 the grammar, making it match a text file in form. There is no way for an application or test suite  
 81087 to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

81088 POSIX.1-2017 requires several changes from historical implementations in order to support  
 81089 internationalization. Probably the most subtle of these is the use of the decimal-point character,  
 81090 defined by the *LC\_NUMERIC* category of the locale, in representations of floating-point  
 81091 numbers. This locale-specific character is used in recognizing numeric input, in converting  
 81092 between strings and numeric values, and in formatting output. However, regardless of locale,  
 81093 the <period> character (the decimal-point character of the POSIX locale) is the decimal-point  
 81094 character recognized in processing *awk* programs (including assignments in command line  
 81095 arguments). This is essentially the same convention as the one used in the ISO C standard. The  
 81096 difference is that the C language includes the *setlocale()* function, which permits an application  
 81097 to modify its locale. Because of this capability, a C application begins executing with its locale set  
 81098 to the C locale, and only executes in the environment-specified locale after an explicit call to  
 81099 *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as  
 81100 inappropriate for POSIX.1-2017. It is possible to execute an *awk* program explicitly in any desired  
 81101 locale by setting the environment in the shell.

81102 The undefined behavior resulting from NULs in extended regular expressions allows future



81103 extensions for the GNU *gawk* program to process binary data.

81104 The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic  
81105 errors) is undefined because it was considered overly limiting on implementations to specify. In  
81106 most cases such errors can be expected to produce a diagnostic and a non-zero exit status.  
81107 However, some implementations may choose to extend the language in ways that make use of  
81108 certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but  
81109 otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect  
81110 in some implementations. Also, different implementations might detect a given error during an  
81111 initial parsing of the program (before reading any input files) while others might detect it when  
81112 executing the program after reading some input. Implementors should be aware that diagnosing  
81113 errors as early as possible and producing useful diagnostics can ease debugging of applications,  
81114 and thus make an implementation more usable.

81115 The unspecified behavior from using multi-character **RS** values is to allow possible future  
81116 extensions based on extended regular expressions used for record separators. Historical  
81117 implementations take the first character of the string and ignore the others.

81118 Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future  
81119 extension that would split up a string into an array of individual characters.

81120 In the context of the **getline** function, equally good arguments for different precedences of the |  
81121 and < operators can be made. Historical practice has been that:

```
81122 getline < "a" "b"
```

81123 is parsed as:

```
81124 (getline < "a") "b"
```

81125 although many would argue that the intent was that the file **ab** should be read. However:

```
81126 getline < "x" + 1
```

81127 parses as:

```
81128 getline < ("x" + 1)
```

81129 Similar problems occur with the | version of **getline**, particularly in combination with \$. For  
81130 example:

```
81131 $"echo hi" | getline
```

81132 (This situation is particularly problematic when used in a **print** statement, where the |**getline**  
81133 part might be a redirection of the **print**.)

81134 Since in most cases such constructs are not (or at least should not) be used (because they have a  
81135 natural ambiguity for which there is no conventional parsing), the meaning of these constructs  
81136 has been made explicitly unspecified. (The effect is that a conforming application that runs into  
81137 the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual  
81138 uses of such constructs.

81139 Grammars can be written that would cause an error under these circumstances. Where  
81140 backwards-compatibility is not a large consideration, implementors may wish to use such  
81141 grammars.

81142 Some historical implementations have allowed some built-in functions to be called without an  
81143 argument list, the result being a default argument list chosen in some ``reasonable'' way. Use of  
81144 **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely  
81145 known or widely used; this particular form is documented in various places (for example, most

81146 historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as  
 81147 legitimate practice. With this exception, default argument lists have always been undocumented  
 81148 and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-  
 81149 defined functions. They add no useful functionality and preclude possible future extensions that  
 81150 might need to name functions without calling them. Not standardizing them seems the simplest  
 81151 course. The standard developers considered that **length** merited special treatment, however,  
 81152 since it has been documented in the past and sees possibly substantial use in historical  
 81153 programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the  
 81154 obsolescent marking for XSI-conforming implementations and many otherwise conforming  
 81155 applications depend on this feature.

81156 In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive  
 81157 <backslash> characters should be used in the string to ensure a single <backslash> will precede  
 81158 the <ampersand> when the resultant string is passed to the function. (For example, to specify  
 81159 one literal <ampersand> in the replacement string, use **gsub(ERE, "\\&")**.)

81160 Historically, the only special character in the *repl* argument of **sub** and **gsub** string functions was  
 81161 the <ampersand> ('&') character and preceding it with the <backslash> character was used to  
 81162 turn off its special meaning.

81163 The description in the ISO POSIX-2:1993 standard introduced behavior such that the  
 81164 <backslash> character was another special character and it was unspecified whether there were  
 81165 any other special characters. This description introduced several portability problems, some of  
 81166 which are described below, and so it has been replaced with the more historical description.  
 81167 Some of the problems include:

81168 Historically, to create the replacement string, a script could use **gsub(ERE, "\\&")**, but  
 81169 with the ISO POSIX-2:1993 standard wording, it was necessary to use **gsub(ERE,**  
 81170 **"\\\\&")**. The <backslash> characters are doubled here because all string literals are  
 81171 subject to lexical analysis, which would reduce each pair of <backslash> characters to a  
 81172 single <backslash> before being passed to **gsub**.

81173 Since it was unspecified what the special characters were, for portable scripts to guarantee  
 81174 that characters are printed literally, each character had to be preceded with a <backslash>.  
 81175 (For example, a portable script had to use **gsub(ERE, "\\h\\i")** to produce a replacement  
 81176 string of "hi".)

81177 The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe  
 81178 historical practice because of the way numeric strings are compared as numbers. The current  
 81179 rules cause the following code:

```
81180 if (0 == "000")
81181 print "strange, but true"
81182 else
81183 print "not true"
```

81184 to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this  
 81185 is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

81186 To fix this problem, the definition of *numeric string* was enhanced to include only those values  
 81187 obtained from specific circumstances (mostly external sources) where it is not possible to  
 81188 determine unambiguously whether the value is intended to be a string or a numeric.

81189 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For  
 81190 example, the notion of a numeric string can be propagated across assignments.) In comparisons,  
 81191 all variables having the uninitialized value are to be treated as a numeric operand evaluating to  
 81192 the numeric value zero.

81193 Uninitialized variables include all types of variables including scalars, array elements, and  
 81194 fields. The definition of an uninitialized value in [Variables and Special Variables](#) (on page 2489)  
 81195 is necessary to describe the value placed on uninitialized variables and on fields that are valid  
 81196 (for example, `< $NF`) but have no characters in them and to describe how these variables are to  
 81197 be used in comparisons. A valid field, such as `$1`, that has no characters in it can be obtained  
 81198 from an input line of `"\t\t"` when `FS='\t'`. Historically, the comparison (`$1<10`) was done  
 81199 numerically after evaluating `$1` to the value zero.

81200 The phrase "... also shall have the numeric value of the numeric string" was removed from  
 81201 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary  
 81202 implementation detail. It is not necessary for POSIX.1-2017 to specify that these objects be  
 81203 assigned two different values. It is only necessary to specify that these objects may evaluate to  
 81204 two different values depending on context.

81205 Historical implementations of *awk* did not parse hexadecimal integer or floating constants like  
 81206 `"0xa"` and `"0xap0"`. Due to an oversight, the 2001 through 2004 editions of this standard  
 81207 required support for hexadecimal floating constants. This was due to the reference to `atof()`.  
 81208 This version of the standard allows but does not require implementations to use `atof()` and  
 81209 includes a description of how floating-point numbers are recognized as an alternative to match  
 81210 historic behavior. The intent of this change is to allow implementations to recognize floating-  
 81211 point constants according to either the ISO/IEC 9899:1990 standard or ISO/IEC 9899:1999  
 81212 standard, and to allow (but not require) implementations to recognize hexadecimal integer  
 81213 constants.

81214 Historical implementations of *awk* did not support floating-point infinities and NaNs in *numeric*  
 81215 *strings*; e.g., `"-INF"` and `"NaN"`. However, implementations that use the `atof()` or `strtod()`  
 81216 functions to do the conversion picked up support for these values if they used a  
 81217 ISO/IEC 9899:1999 standard version of the function instead of a ISO/IEC 9899:1990 standard  
 81218 version. Due to an oversight, the 2001 through 2004 editions of this standard did not allow  
 81219 support for infinities and NaNs, but in this revision support is allowed (but not required). This is  
 81220 a silent change to the behavior of *awk* programs; for example, in the POSIX locale the expression:

81221 `("-INF" + 0 < 0)`

81222 formerly had the value 0 because `"-INF"` converted to 0, but now it may have the value 0 or 1.

#### 81223 FUTURE DIRECTIONS

81224 A future version of this standard may require the `!="` and `=="` operators to perform string  
 81225 comparisons by checking if the strings are identical (and not by checking if they collate equally).

#### 81226 SEE ALSO

81227 [Section 1.3](#) (on page 2335), [grep](#), [lex](#), [sed](#)

81228 [XBD Chapter 5](#) (on page 121), [Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Chapter 9](#) (on  
 81229 page 181), [Section 12.2](#) (on page 216)

81230 [XSH `atof\(\)`](#), [exec](#), [isspace\(\)](#), [popen\(\)](#), [setlocale\(\)](#), [strtod\(\)](#)

#### 81231 CHANGE HISTORY

81232 First released in Issue 2.

#### 81233 Issue 5

81234 The FUTURE DIRECTIONS section is added.

81235 **Issue 6**

81236 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

81237 The normative text is reworded to avoid use of the term “must” for application requirements.

81238 IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence “An occurrence of two  
81239 consecutive <backslash> characters shall be interpreted as just a single literal <backslash>  
81240 character.” into the description of the **sub** string function.

81241 **Issue 7**

81242 PASC Interpretation 1003.2-1992 #107 (SD5-XCU-ERN-73) is applied, updating the description of  
81243 the **OFS** variable.

81244 Austin Group Interpretation 1003.1-2001 #189 is applied.

81245 Austin Group Interpretation 1003.1-2001 #201 is applied, permitting implementations to support  
81246 infinities and NaNs.

81247 SD5-XCU-ERN-79 is applied, restoring the horizontal lines to [Table 4-1](#) (on page 2485), and  
81248 SD5-XCU-ERN-80 is applied, changing the order of some table entries.

81249 SD5-XCU-ERN-87 is applied, updating the descriptive text of the Grammar.

81250 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81251 The EXTENDED DESCRIPTION is changed to make the support of hexadecimal integer and  
81252 floating constants optional.

81253 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0057 [224], XCU/TC1-2008/0058  
81254 [454], XCU/TC1-2008/0059 [224], XCU/TC1-2008/0060 [224], XCU/TC1-2008/0061 [254],  
81255 XCU/TC1-2008/0062 [254], XCU/TC1-2008/0063 [224], and XCU/TC1-2008/0064 [454] are  
81256 applied.

81257 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0058 [584], XCU/TC2-2008/0059  
81258 [963], XCU/TC2-2008/0060 [226], XCU/TC2-2008/0061 [663], XCU/TC2-2008/0062 [963],  
81259 XCU/TC2-2008/0063 [226], and XCU/TC2-2008/0064 [963] are applied.

81260 **NAME**

81261 `basename` — return non-directory portion of a pathname

81262 **SYNOPSIS**

81263 `basename string [suffix]`

81264 **DESCRIPTION**

81265 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.271](#) (on page 76).  
 81266 The string *string* shall be converted to the filename corresponding to the last pathname  
 81267 component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be  
 81268 done by performing actions equivalent to the following steps in order:

- 81269 1. If *string* is a null string, it is unspecified whether the resulting string is '.' or a null  
 81270 string. In either case, skip steps 2 through 6.
- 81271 2. If *string* is "/", it is implementation-defined whether steps 3 to 6 are skipped or  
 81272 processed.
- 81273 3. If *string* consists entirely of <slash> characters, *string* shall be set to a single <slash>  
 81274 character. In this case, skip steps 4 to 6.
- 81275 4. If there are any trailing <slash> characters in *string*, they shall be removed.
- 81276 5. If there are any <slash> characters remaining in *string*, the prefix of *string* up to and  
 81277 including the last <slash> character in *string* shall be removed.
- 81278 6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is  
 81279 identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed  
 81280 from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an  
 81281 error if *suffix* is not found in *string*.

81282 The resulting string shall be written to standard output.

81283 **OPTIONS**

81284 None.

81285 **OPERANDS**

81286 The following operands shall be supported:

81287 *string* A string.

81288 *suffix* A string.

81289 **STDIN**

81290 Not used.

81291 **INPUT FILES**

81292 None.

81293 **ENVIRONMENT VARIABLES**

81294 The following environment variables shall affect the execution of *basename*:

81295 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 81296 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 81297 variables used to determine the values of locale categories.)

81298 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 81299 internationalization variables.

81300 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 81301 characters (for example, single-byte as opposed to multi-byte characters in  
 81302 arguments).

81303 **LC\_MESSAGES**  
 81304 Determine the locale that should be used to affect the format and contents of  
 81305 diagnostic messages written to standard error.

81306 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

81307 **ASYNCHRONOUS EVENTS**  
 81308 Default.

81309 **STDOUT**  
 81310 The *basename* utility shall write a line to the standard output in the following format:  
 81311 "%s\n", <resulting string>

81312 **STDERR**  
 81313 The standard error shall be used only for diagnostic messages.

81314 **OUTPUT FILES**  
 81315 None.

81316 **EXTENDED DESCRIPTION**  
 81317 None.

81318 **EXIT STATUS**  
 81319 The following exit values shall be returned:  
 81320 0 Successful completion.  
 81321 >0 An error occurred.

81322 **CONSEQUENCES OF ERRORS**  
 81323 Default.

81324 **APPLICATION USAGE**  
 81325 The definition of *pathname* specifies implementation-defined behavior for pathnames starting  
 81326 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters  
 81327 to the beginning of a pathname unless they can ensure that there are more or less than two or are  
 81328 prepared to deal with the implementation-defined consequences.

81329 **EXAMPLES**  
 81330 If the string *string* is a valid pathname:  
 81331 `$(basename -- "string")`  
 81332 produces a filename that could be used to open the file named by *string* in the directory returned  
 81333 by:  
 81334 `$(dirname -- "string")`  
 81335 If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be  
 81336 a valid filename. The *basename* utility is not expected to make any judgements about the validity  
 81337 of *string* as a pathname; it just follows the specified algorithm to produce a result string.

81338 The following shell script compiles `/usr/src/cmd/cat.c` and moves the output to a file named `cat`  
 81339 in the current directory when invoked with the argument `/usr/src/cmd/cat` or with the argument  
 81340 `/usr/src/cmd/cat.c`:  
 81341 `c99 -- "$(dirname -- "$1")/$(basename -- "$1" .c).c" &&`  
 81342 `mv a.out "$(basename -- "$1" .c)"`

81343 The EXAMPLES section of the *basename()* function (see XSH *basename()*) includes a table  
 81344 showing examples of the results of processing several sample pathnames by the *basename()* and

81345 *dirname()* functions and by the *basename* and *dirname* utilities.

#### 81346 RATIONALE

81347 The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid  
81348 pathname:

```
81349 $(basename -- "string")
```

81350 would be a valid filename for the file in the directory:

```
81351 $(dirname -- "string")
```

81352 This would not work for the early proposal versions of these utilities due to the way it specified  
81353 handling of trailing <slash> characters.

81354 Since the definition of *pathname* specifies implementation-defined behavior for pathnames  
81355 starting with two <slash> characters, this volume of POSIX.1-2017 specifies similar  
81356 implementation-defined behavior for the *basename* and *dirname* utilities.

#### 81357 FUTURE DIRECTIONS

81358 None.

#### 81359 SEE ALSO

81360 [Section 2.5](#) (on page 2349), *dirname*

81361 XBD [Section 3.271](#) (on page 76), [Chapter 8](#) (on page 173)

81362 XSH *basename()*, *dirname()*

#### 81363 CHANGE HISTORY

81364 First released in Issue 2.

#### 81365 Issue 6

81366 IEEE PASC Interpretation 1003.2 #164 is applied.

81367 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 81368 Issue 7

81369 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0065 [192,538], XCU/TC1-2008/0066  
81370 [192,538], and XCU/TC1-2008/0067 [192,430,538] are applied.

81371 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0065 [612] is applied.

81372 **NAME**81373 `batch` ‡schedule commands to be executed in a batch queue81374 **SYNOPSIS**81375 `batch`81376 **DESCRIPTION**81377 The `batch` utility shall read commands from standard input and schedule them for execution in a  
81378 batch queue. It shall be the equivalent of the command:81379 `at -q b -m now`81380 where queue `b` is a special `at` queue, specifically for batch jobs. Batch jobs shall be submitted to  
81381 the batch queue with no time constraints and shall be run by the system using algorithms, based  
81382 on unspecified factors, that may vary with each invocation of `batch`.81383 XSI Users shall be permitted to use `batch` if their name appears in the file `at.allow` which is located in  
81384 an implementation-defined directory. If that file does not exist, the file `at.deny`, which is located  
81385 in an implementation-defined directory, shall be checked to determine whether the user shall be  
81386 denied access to `batch`. If neither file exists, only a process with appropriate privileges shall be  
81387 allowed to submit a job. If only `at.deny` exists and is empty, global usage shall be permitted. The  
81388 `at.allow` and `at.deny` files shall consist of one user name per line.81389 **OPTIONS**

81390 None.

81391 **OPERANDS**

81392 None.

81393 **STDIN**81394 The standard input shall be a text file consisting of commands acceptable to the shell command  
81395 language described in [Chapter 2](#) (on page 2345).81396 **INPUT FILES**81397 XSI The text files `at.allow` and `at.deny`, which are located in an implementation-defined directory,  
81398 shall contain zero or more user names, one per line, of users who are, respectively, authorized or  
81399 denied access to the `at` and `batch` utilities.81400 **ENVIRONMENT VARIABLES**81401 The following environment variables shall affect the execution of `batch`:81402 `LANG` Provide a default value for the internationalization variables that are unset or null.  
81403 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
81404 variables used to determine the values of locale categories.)81405 `LC_ALL` If set to a non-empty string value, override the values of all the other  
81406 internationalization variables.81407 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
81408 characters (for example, single-byte as opposed to multi-byte characters in  
81409 arguments and input files).81410 `LC_MESSAGES`81411 Determine the locale that should be used to affect the format and contents of  
81412 diagnostic messages written to standard error and informative messages written to  
81413 standard output.81414 `LC_TIME` Determine the format and contents for date and time strings written by `batch`.



|       |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------|-----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 81415 | XSI | <b>NLSPATH</b>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                                                                                                                            |
| 81416 |     | <b>SHELL</b>                  | Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen. |
| 81417 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81418 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81419 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81420 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81421 |     | <b>TZ</b>                     | Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it overrides <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.                        |
| 81422 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81423 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81424 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81425 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81426 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81427 |     |                               | Default.                                                                                                                                                                                                                                                                                                                                                                                                         |
| 81428 |     | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81429 |     |                               | When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.                                                                                                                                                                                                                                               |
| 81430 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81431 |     | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81432 |     |                               | The following shall be written to standard error when a job has been successfully submitted:                                                                                                                                                                                                                                                                                                                     |
| 81433 |     |                               | "job %s at %s\n", <i>at_job_id</i> , < <i>date</i> >                                                                                                                                                                                                                                                                                                                                                             |
| 81434 |     |                               | where <i>date</i> shall be equivalent in format to the output of:                                                                                                                                                                                                                                                                                                                                                |
| 81435 |     |                               | date +"%a %b %e %T %Y"                                                                                                                                                                                                                                                                                                                                                                                           |
| 81436 |     |                               | The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the <i>TZ</i> variable).                                                                                                                                                                                                                                                                           |
| 81437 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81438 |     |                               | Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.                                                                                                                                                                                                                                                                |
| 81439 |     |                               |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81440 |     |                               | Diagnostic messages, if any, shall be written to standard error.                                                                                                                                                                                                                                                                                                                                                 |
| 81441 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81442 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 81443 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81444 |     |                               | None.                                                                                                                                                                                                                                                                                                                                                                                                            |
| 81445 |     | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81446 |     |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                                                                                                                                     |
| 81447 |     |                               | 0 Successful completion.                                                                                                                                                                                                                                                                                                                                                                                         |
| 81448 |     |                               | >0 An error occurred.                                                                                                                                                                                                                                                                                                                                                                                            |
| 81449 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 81450 |     |                               | The job shall not be scheduled.                                                                                                                                                                                                                                                                                                                                                                                  |

81451 **APPLICATION USAGE**

81452 It may be useful to redirect standard output within the specified commands.

81453 **EXAMPLES**

81454 1. This sequence can be used at a terminal:

```
81455 batch
81456 sort < file >outfile
81457 EOT
```

81458 2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a  
81459 command procedure (the sequence of output redirection specifications is significant):

```
81460 batch <<!
81461 diff file1 file2 2>&1 >outfile | mailx mygroup
81462 !
```

81463 **RATIONALE**

81464 Early proposals described *batch* in a manner totally separated from *at*, even though the historical  
81465 model treated it almost as a synonym for *at -qb*. A number of features were added to list and  
81466 control batch work separately from those in *at*. Upon further reflection, it was decided that the  
81467 benefit of this did not merit the change to the historical interface.

81468 The *-m* option was included on the equivalent *at* command because it is historical practice to  
81469 mail results to the submitter, even if all job-produced output is redirected. As explained in the  
81470 RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling  
81471 delays), despite some historical systems where *at now* would have been considered an error.

81472 **FUTURE DIRECTIONS**

81473 None.

81474 **SEE ALSO**81475 *at*81476 XBD [Chapter 8](#) (on page 173)81477 **CHANGE HISTORY**

81478 First released in Issue 2.

81479 **Issue 6**

81480 This utility is marked as part of the User Portability Utilities option.

81481 The NAME is changed to align with the IEEE P1003.2b draft standard.

81482 The normative text is reworded to avoid use of the term “must” for application requirements.

81483 **Issue 7**

81484 The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability  
81485 Utilities is now an option for interactive utilities.

81486 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced  
81487 by the *batch* utility.

81488 **NAME**

81489 bc — arbitrary-precision arithmetic language

81490 **SYNOPSIS**81491 bc [-l] [*file...*]81492 **DESCRIPTION**

81493 The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files  
 81494 given, then read from the standard input. If the standard input and standard output to *bc* are  
 81495 attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing  
 81496 behavioral constraints described in the following sections.

81497 **OPTIONS**81498 The *bc* utility shall conform to XBD [Section 12.2](#) (on page 216).

81499 The following option shall be supported:

81500 **-l** (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the  
 81501 default zero; see the EXTENDED DESCRIPTION section.

81502 **OPERANDS**

81503 The following operand shall be supported:

81504 *file* A pathname of a text file containing *bc* program statements. After all *files* have  
 81505 been read, *bc* shall read the standard input.

81506 **STDIN**

81507 See the INPUT FILES section.

81508 **INPUT FILES**

81509 Input files shall be text files containing a sequence of comments, statements, and function  
 81510 definitions that shall be executed as they are read.

81511 **ENVIRONMENT VARIABLES**81512 The following environment variables shall affect the execution of *bc*:

81513 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 81514 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization  
 81515 variables used to determine the values of locale categories.)

81516 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 81517 internationalization variables.

81518 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 81519 characters (for example, single-byte as opposed to multi-byte characters in  
 81520 arguments and input files).

81521 *LC\_MESSAGES*

81522 Determine the locale that should be used to affect the format and contents of  
 81523 diagnostic messages written to standard error.

81524 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.81525 **ASYNCHRONOUS EVENTS**

81526 Default.

81527 **STDOUT**

81528 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more  
 81529 lines containing the value of all executed expressions without assignments. The radix and  
 81530 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the  
 81531 EXTENDED DESCRIPTION section.

81532 **STDERR**

81533 The standard error shall be used only for diagnostic messages.

81534 **OUTPUT FILES**

81535 None.

81536 **EXTENDED DESCRIPTION**81537 **Grammar**

81538 The grammar in this section and the lexical conventions in the following section shall together  
 81539 describe the syntax for *bc* programs. The general conventions for this style of grammar are  
 81540 described in [Section 1.3](#) (on page 2335). A valid program can be represented as the non-terminal  
 81541 symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax  
 81542 description.

```

81543 %token EOF NEWLINE STRING LETTER NUMBER
81544 %token MUL_OP
81545 /* '*' , '/' , '%' */
81546 %token ASSIGN_OP
81547 /* '=' , '+=' , '-=' , '*=' , '/=' , '%=' , '^=' */
81548 %token REL_OP
81549 /* '==' , '<=' , '>=' , '!=' , '<' , '>' */
81550 %token INCR_DECR
81551 /* '++' , '--' */
81552 %token Define Break Quit Length
81553 /* 'define' , 'break' , 'quit' , 'length' */
81554 %token Return For If While Sqrt
81555 /* 'return' , 'for' , 'if' , 'while' , 'sqrt' */
81556 %token Scale Ibase Obase Auto
81557 /* 'scale' , 'ibase' , 'obase' , 'auto' */
81558 %start program
81559 %%
81560 program : EOF
81561 | input_item program
81562 ;
81563 input_item : semicolon_list NEWLINE
81564 | function
81565 ;
81566 semicolon_list : /* empty */
81567 | statement
81568 | semicolon_list ';' statement
81569 | semicolon_list ';'
81570 ;
81571 statement_list : /* empty */
81572 | statement
81573 | statement_list NEWLINE

```

```

81574 | statement_list NEWLINE statement
81575 | statement_list ';'
81576 | statement_list ';' statement
81577 ;
81578 statement : expression
81579 | STRING
81580 | Break
81581 | Quit
81582 | Return
81583 | Return '(' return_expression ')'
81584 | For '(' expression ';'
81585 relational_expression ';'
81586 expression ')' statement
81587 | If '(' relational_expression ')' statement
81588 | While '(' relational_expression ')' statement
81589 | '{' statement_list '}'
81590 ;
81591 function : Define LETTER '(' opt_parameter_list ')'
81592 '{' NEWLINE opt_auto_define_list
81593 statement_list '}'
81594 ;
81595 opt_parameter_list : /* empty */
81596 | parameter_list
81597 ;
81598 parameter_list : LETTER
81599 | define_list ',' LETTER
81600 ;
81601 opt_auto_define_list : /* empty */
81602 | Auto define_list NEWLINE
81603 | Auto define_list ';'
81604 ;
81605 define_list : LETTER
81606 | LETTER '[' ']'
81607 | define_list ',' LETTER
81608 | define_list ',' LETTER '[' ']'
81609 ;
81610 opt_argument_list : /* empty */
81611 | argument_list
81612 ;
81613 argument_list : expression
81614 | LETTER '[' ']' ',' argument_list
81615 ;
81616 relational_expression : expression
81617 | expression REL_OP expression
81618 ;
81619 return_expression : /* empty */
81620 | expression

```

```

81621 ;
81622 expression : named_expression
81623 | NUMBER
81624 | '(' expression ')'
81625 | LETTER '(' opt_argument_list ')'
81626 | '-' expression
81627 | expression '+' expression
81628 | expression '-' expression
81629 | expression MUL_OP expression
81630 | expression '^' expression
81631 | INCR_DECR named_expression
81632 | named_expression INCR_DECR
81633 | named_expression ASSIGN_OP expression
81634 | Length '(' expression ')'
81635 | Sqrt '(' expression ')'
81636 | Scale '(' expression ')'
81637 ;
81638 named_expression : LETTER
81639 | LETTER '[' expression ']'
81640 | Scale
81641 | Ibase
81642 | Obase
81643 ;

```

#### 81644 Lexical Conventions in bc

81645 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as  
 81646 follows:

- 81647 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a  
 81648 given point.
- 81649 2. A comment shall consist of any characters beginning with the two adjacent characters  
 81650 `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`.  
 81651 Comments shall have no effect except to delimit lexical tokens.
- 81652 3. The `<newline>` shall be recognized as the token **NEWLINE**.
- 81653 4. The token **STRING** shall represent a string constant; it shall consist of any characters  
 81654 beginning with the double-quote character (`"`) and terminated by another occurrence  
 81655 of the double-quote character. The value of the string is the sequence of all characters  
 81656 between, but not including, the two double-quote characters. All characters shall be taken  
 81657 literally from the input, and there is no way to specify a string containing a double-quote  
 81658 character. The length of the value of each string shall be limited to `{BC_STRING_MAX}`  
 81659 bytes.
- 81660 5. A `<blank>` shall have no effect except as an ordinary character if it appears within a  
 81661 **STRING** token, or to delimit a lexical token other than **STRING**.
- 81662 6. The combination of a `<backslash>` character immediately followed by a `<newline>` shall  
 81663 have no effect other than to delimit lexical tokens with the following exceptions:

- 81664                   It shall be interpreted as the character sequence "\<newline>" in **STRING** tokens.
- 81665                   It shall be ignored as part of a multi-line **NUMBER** token.
- 81666           7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the  
81667 following grammar:
- ```
81668           NUMBER : integer
81669                   | '.' integer
81670                   | integer '.'
81671                   | integer '.' integer
81672                   ;
```
- ```
81673 integer : digit
81674 | integer digit
81675 ;
```
- ```
81676           digit  : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
81677                   | 8 | 9 | A | B | C | D | E | F
81678                   ;
```
- 81679 8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by
81680 the value of the internal register **ibase** (described below). Each of the **digit** characters
81681 shall have the value from 0 to 15 in the order listed here, and the <period> character shall
81682 represent the radix point. The behavior is undefined if digits greater than or equal to the
81683 value of **ibase** appear in the token. However, note the exception for single-digit values
81684 being assigned to **ibase** and **obase** themselves, in [Operations in bc](#) (on page 2529).
- 81685 9. The following keywords shall be recognized as tokens:
- ```
81686 auto ibase length return while
81687 break if obase scale
81688 define for quit sqrt
```
- 81689           10. Any of the following characters occurring anywhere except within a keyword shall be  
81690 recognized as the token **LETTER**:
- ```
81691           a b c d e f g h i j k l m n o p q r s t u v w x y z
```
- 81692 11. The following single-character and two-character sequences shall be recognized as the
81693 token **ASSIGN_OP**:
- ```
81694 = += -= *= /= %= ^=
```
- 81695           12. If an '=' character, as the beginning of a token, is followed by a '-' character with no  
81696 intervening delimiter, the behavior is undefined.
- 81697           13. The following single-characters shall be recognized as the token **MUL\_OP**:
- ```
81698           *   /   %
```
- 81699 14. The following single-character and two-character sequences shall be recognized as the
81700 token **REL_OP**:
- ```
81701 == <= >= != < >
```
- 81702           15. The following two-character sequences shall be recognized as the token **INCR\_DECR**:
- ```
81703           ++   --
```

81704 16. The following single characters shall be recognized as tokens whose names are the
81705 character:

81706 <newline> () , + - ; [] ^ { }

81707 17. The token **EOF** is returned when the end of input is reached.

81708 Operations in bc

81709 There are three kinds of identifiers: ordinary identifiers, array identifiers, and function
81710 identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed
81711 by square brackets ("[]"). An array subscript is required except in an argument or auto list.
81712 Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing shall
81713 begin at zero so an array is indexed from 0 to {BC_DIM_MAX}-1. Subscripts shall be truncated
81714 to integers. The application shall ensure that function identifiers are followed by parentheses,
81715 possibly enclosing arguments. The three types of identifiers do not conflict.

81716 The following table summarizes the rules for precedence and associativity of all operators.
81717 Operators on the same line shall have the same precedence; rows are in order of decreasing
81718 precedence.

81719 **Table 4-3** Operators in *bc*

81720	Operator	Associativity
81721	++, --	N/A
81722	unary -	N/A
81723	^	Right to left
81724	*, /, %	Left to right
81725	+, binary -	Left to right
81726	=, +=, -=, *=, /=, %=, ^=	Right to left
81727	==, <=, >=, !=, <, >	None

81728 Each expression or named expression has a *scale*, which is the number of decimal digits that
81729 shall be maintained as the fractional portion of the expression.

81730 *Named expressions* are places where values are stored. Named expressions shall be valid on the
81731 left side of an assignment. The value of a named expression shall be the value stored in the place
81732 named. Simple identifiers and array elements are named expressions; they have an initial value
81733 of zero and an initial scale of zero.

81734 The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an
81735 expression consisting of the name of one of these registers shall be zero; values assigned to any
81736 of these registers are truncated to integers. The **scale** register shall contain a global value used in
81737 computing the scale of expressions (as described below). The value of the register **scale** is
81738 limited to $0 \leq \text{scale} \leq \{\text{BC_SCALE_MAX}\}$ and shall have a default value of zero. The **ibase** and
81739 **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be
81740 limited to:

81741 $2 \leq \text{ibase} \leq 16$

81742 The value of **obase** shall be limited to:

81743 $2 \leq \text{obase} \leq \{\text{BC_BASE_MAX}\}$

81744 When either **ibase** or **obase** is assigned a single **digit** value from the list in [Lexical Conventions](#)
81745 [in bc](#) (on page 2527), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to
81746 base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when

81747 digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall
81748 have initial values of 10.

81749 Internal computations shall be conducted as if in decimal, regardless of the input and output
81750 bases, to the specified number of decimal digits. When an exact result is not achieved (for
81751 example, **scale=0**; 3.2/1), the result shall be truncated.

81752 For all values of **obase** specified by this volume of POSIX.1-2017, *bc* shall output numeric values
81753 by performing each of the following steps in order:

- 81754 1. If the value is less than zero, a <hyphen-minus> ('-') character shall be output.
- 81755 2. One of the following is output, depending on the numerical value:
 - 81756 If the absolute value of the numerical value is greater than or equal to one, the
81757 integer portion of the value shall be output as a series of digits appropriate to **obase**
81758 (as described below), most significant digit first. The most significant non-zero digit
81759 shall be output next, followed by each successively less significant digit.
 - 81760 If the absolute value of the numerical value is less than one but greater than zero
81761 and the scale of the numerical value is greater than zero, it is unspecified whether
81762 the character 0 is output.
 - 81763 If the numerical value is zero, the character 0 shall be output.
- 81764 3. If the scale of the value is greater than zero and the numeric value is not zero, a <period>
81765 character shall be output, followed by a series of digits appropriate to **obase** (as described
81766 below) representing the most significant portion of the fractional part of the value. If *s*
81767 represents the scale of the value being output, the number of digits output shall be *s* if
81768 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s*
81769 if **obase** is less than 10. For **obase** values other than 10, this should be the number of
81770 digits needed to represent a precision of 10^{*s*}.

81771 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

81772 0 1 2 3 4 5 6 7 8 9 A B C D E F

81773 which represent the values zero to 15, inclusive, respectively.

81774 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.
81775 Each digit except the most significant fractional digit shall be preceded by a single <space>. For
81776 bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1000, three-
81777 digit decimal strings, and so on. For example, the decimal number 1024 in base 25 would be
81778 written as:

81779 Δ01Δ15Δ24

81780 and in base 125, as:

81781 Δ008Δ024

81782 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;
81783 other locales may split at different character boundaries. Lines that are continued shall end with
81784 a <backslash>.

81785 A function call shall consist of a function name followed by parentheses containing a
81786 <comma>-separated list of expressions, which are the function arguments. A whole array
81787 passed as an argument shall be specified by the array name followed by empty square brackets.
81788 All function arguments shall be passed by value. As a result, changes made to the formal
81789 parameters shall have no effect on the actual arguments. If the function terminates by executing

81790 a **return** statement, the value of the function shall be the value of the expression in the
81791 parentheses of the **return** statement or shall be zero if no expression is provided or if there is no
81792 **return** statement.

81793 The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be
81794 truncated in the least significant decimal place. The scale of the result shall be the scale of the
81795 expression or the value of **scale**, whichever is larger.

81796 The result of **length**(*expression*) shall be the total number of significant decimal digits in the
81797 expression. The scale of the result shall be zero.

81798 The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be
81799 zero.

81800 A numeric constant shall be an expression. The scale shall be the number of digits that follow the
81801 radix point in the input representing the constant, or zero if no radix point appears.

81802 The sequence (*expression*) shall be an expression with the same value and scale as *expression*.
81803 The parentheses can be used to alter the normal precedence.

81804 The semantics of the unary and binary operators are as follows:

81805 *-expression*
81806 The result shall be the negative of the *expression*. The scale of the result shall be the scale of
81807 *expression*.

81808 The unary increment and decrement operators shall not modify the scale of the named
81809 expression upon which they operate. The scale of the result shall be the scale of that named
81810 expression.

81811 *++named-expression*
81812 The named expression shall be incremented by one. The result shall be the value of the
81813 named expression after incrementing.

81814 *--named-expression*
81815 The named expression shall be decremented by one. The result shall be the value of the
81816 named expression after decrementing.

81817 *named-expression++*
81818 The named expression shall be incremented by one. The result shall be the value of the
81819 named expression before incrementing.

81820 *named-expression--*
81821 The named expression shall be decremented by one. The result shall be the value of the
81822 named expression before decrementing.

81823 The exponentiation operator, <circumflex> ('^'), shall bind right to left.

81824 *expression^expression*
81825 The result shall be the first *expression* raised to the power of the second *expression*. If the
81826 second expression is not an integer, the behavior is undefined. If *a* is the scale of the left
81827 expression and *b* is the absolute value of the right expression, the scale of the result shall be:
81828 if $b \geq 0$ $\min(a * b, \max(\text{scale}, a))$ if $b < 0$ *scale*

81829 The multiplicative operators ('*', '/', '%') shall bind left to right.

81830 *expression*expression*
81831 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two
81832 expressions, then the scale of the result shall be:

81833 $\min(a+b, \max(\text{scale}, a, b))$

81834 *expression/expression*

81835 The result shall be the quotient of the two expressions. The scale of the result shall be the
81836 value of **scale**.

81837 *expression%expression*

81838 For expressions *a* and *b*, *a%b* shall be evaluated equivalent to the steps:

81839 1. Compute *a/b* to current scale.

81840 2. Use the result to compute:

81841 $a - (a / b) * b$

81842 to scale:

81843 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

81844 The scale of the result shall be:

81845 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

81846 When **scale** is zero, the '%' operator is the mathematical remainder operator.

81847 The additive operators ('+', '-') shall bind left to right.

81848 *expression+expression*

81849 The result shall be the sum of the two expressions. The scale of the result shall be the
81850 maximum of the scales of the expressions.

81851 *expression-expression*

81852 The result shall be the difference of the two expressions. The scale of the result shall be the
81853 maximum of the scales of the expressions.

81854 The assignment operators ('=', '+=', '-=', '*=', '/=', '%=', '^=') shall bind right to left.

81855 *named-expression=expression*

81856 This expression shall result in assigning the value of the expression on the right to the
81857 named expression on the left. The scale of both the named expression and the result shall be
81858 the scale of *expression*.

81859 The compound assignment forms:

81860 *named-expression <operator>= expression*

81861 shall be equivalent to:

81862 *named-expression=named-expression <operator> expression*

81863 except that the *named-expression* shall be evaluated only once.

81864 Unlike all other operators, the relational operators ('<', '>', '<=', '>=', '==', '!=') shall be
81865 only valid as the object of an **if**, **while**, or inside a **for** statement.

81866 *expression1<expression2*

81867 The relation shall be true if the value of *expression1* is strictly less than the value of
81868 *expression2*.

81869 *expression1>expression2*

81870 The relation shall be true if the value of *expression1* is strictly greater than the value of
81871 *expression2*.

81872 *expression1* <= *expression2*
 81873 The relation shall be true if the value of *expression1* is less than or equal to the value of
 81874 *expression2*.

81875 *expression1* >= *expression2*
 81876 The relation shall be true if the value of *expression1* is greater than or equal to the value of
 81877 *expression2*.

81878 *expression1* = *expression2*
 81879 The relation shall be true if the values of *expression1* and *expression2* are equal.

81880 *expression1* != *expression2*
 81881 The relation shall be true if the values of *expression1* and *expression2* are unequal.

81882 There are only two storage classes in *bc*: global and automatic (local). Only identifiers that are
 81883 local to a function need be declared with the **auto** command. The arguments to a function shall
 81884 be local to the function. All other identifiers are assumed to be global and available to all
 81885 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto
 81886 shall be allocated on entry to the function and released on returning from the function. They
 81887 therefore do not retain values between function calls. Auto arrays shall be specified by the array
 81888 name followed by empty square brackets. On entry to a function, the old values of the names
 81889 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the
 81890 function returns, reference to these names shall refer only to the new values.

81891 References to any of these names from other functions that are called from this function also
 81892 refer to the new value until one of those functions uses the same name for a local variable.

81893 When a statement is an expression, unless the main operator is an assignment, execution of the
 81894 statement shall write the value of the expression followed by a <newline>.

81895 When a statement is a string, execution of the statement shall write the value of the string.

81896 Statements separated by <semicolon> or <newline> characters shall be executed sequentially. In
 81897 an interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical
 81898 production:

```
81899 input_item : semicolon_list NEWLINE
```

81900 the sequential list of statements making up the **semicolon_list** shall be executed immediately
 81901 and any output produced by that execution shall be written without any delay due to buffering.

81902 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

81903 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;
 81904 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the
 81905 *relation* is false, execution shall resume after *statement*.

81906 A **for** statement (**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

```
81907 first-expression
81908 while (relation) {
81909     statement
81910     last-expression
81911 }
```

81912 The application shall ensure that all three expressions are present.

81913 The **break** statement shall cause termination of a **for** or **while** statement.

81914 The **auto** statement (**auto** *identifier* [*,identifier*] ...) shall cause the values of the identifiers to be

81915 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers
 81916 shall be specified by following the array name by empty square brackets. The application shall
 81917 ensure that the **auto** statement is the first statement in a function definition.

81918 A **define** statement:

```
81919 define LETTER ( opt_parameter_list ) {
81920     opt_auto_define_list
81921     statement_list
81922 }
```

81923 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the
 81924 **define** statement shall replace the previous definition. The expression:

```
81925 LETTER ( opt_argument_list )
```

81926 shall invoke the function named **LETTER**. The behavior is undefined if the number of
 81927 arguments in the invocation does not match the number of parameters in the definition.
 81928 Functions shall be defined before they are invoked. A function shall be considered to be defined
 81929 within its own body, so recursive calls are valid. The values of numeric constants within a
 81930 function shall be interpreted in the base specified by the value of the **ibase** register when the
 81931 function is invoked.

81932 The **return** statements (**return** and **return(expression)**) shall cause termination of a function,
 81933 popping of its auto variables, and specification of the result of the function. The first form shall
 81934 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the
 81935 value and scale of the expression returned.

81936 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement
 81937 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

81938 The following functions shall be defined when the **-I** option is specified:

```
81939 s( expression )
81940     Sine of argument in radians.
```

```
81941 c( expression )
81942     Cosine of argument in radians.
```

```
81943 a( expression )
81944     Arctangent of argument.
```

```
81945 l( expression )
81946     Natural logarithm of argument.
```

```
81947 e( expression )
81948     Exponential function of argument.
```

```
81949 j( expression1, expression2 )
81950     Bessel function of expression2 of the first kind of integer order expression1.
```

81951 The scale of the result returned by these functions shall be the value of the **scale** register at the
 81952 time the function is invoked. The value of the **scale** register after these functions have completed
 81953 their execution shall be the same value it had upon invocation. The behavior is undefined if any
 81954 of these functions is invoked with an argument outside the domain of the mathematical
 81955 function.

81956 **EXIT STATUS**

81957 The following exit values shall be returned:

81958 0 All input files were processed successfully.

81959 *unspecified* An error occurred.81960 **CONSEQUENCES OF ERRORS**81961 If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.81963 In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.81966 **APPLICATION USAGE**81967 Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.81968 For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.81972 The *bc* utility always uses the <period> ('.') character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the <period> character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a <comma> as the decimal-point character:81979

```
define f(a,b) {
81980     ...
81981 }
81982 ...
81983 f(1,2,3)
```

81984 Because of such ambiguities, the <period> character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the <period> is also used in output.

81987 **EXAMPLES**81988 In the shell, the following assigns an approximation of the first ten digits of ' π ' to the variable *x*:81989

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

81990 The following *bc* program prints the same approximation of ' π ', with a label, to standard output:81992

```
scale = 10
81993 "pi equals "
81994 104348 / 33215
```

81995 The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the `-I` option is specified):81997

```
scale = 20
81998 define e(x){
81999     auto a, b, c, i, s
```

```

82000     a = 1
82001     b = 1
82002     s = 1
82003     for (i = 1; 1 == 1; i++){
82004         a = a*x
82005         b = b*i
82006         c = a/b
82007         if (c == 0) {
82008             return(s)
82009         }
82010         s = s+c
82011     }
82012 }

```

82013 The following prints approximate values of the exponential function of the first ten integers:

```

82014 for (i = 1; i <= 10; ++i) {
82015     e(i)
82016 }

```

82017 RATIONALE

82018 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to
82019 be part of this volume of POSIX.1-2017 because *bc* was thought to have a more intuitive
82020 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be
82021 compliant.

82022 The exit status for error conditions has been left unspecified for several reasons:

82023 The *bc* utility is used in both interactive and non-interactive situations. Different exit codes
82024 may be appropriate for the two uses.

82025 It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions,
82026 and syntax errors are all possibilities.

82027 It is not clear what utility the exit status has.

82028 In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with
82029 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*
82030 aborted.

82031 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief
82032 that *bc file1 file2* is used most often when at least *file1* contains data/function
82033 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not
82034 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check
82035 all its files for accessibility before opening any of them.

82036 There was considerable debate on the appropriateness of the language accepted by *bc*. Several
82037 reviewers preferred to see either a pure subset of the C language or some changes to make the
82038 language more compatible with C. While the *bc* language has some obvious similarities to C, it
82039 has never claimed to be compatible with any version of C. An interpreter for a subset of C might
82040 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility
82041 is known in historical practice, and it was not within the scope of this volume of POSIX.1-2017 to
82042 define such a language and utility. If and when they are defined, it may be appropriate to
82043 include them in a future version of this standard. This left the following alternatives:

82044 1. Exclude any calculator language from this volume of POSIX.1-2017.
 82045 The consensus of the standard developers was that a simple programmatic calculator
 82046 language is very useful for both applications and interactive users. The only arguments
 82047 for excluding any calculator were that it would become obsolete if and when a C-
 82048 compatible one emerged, or that the absence would encourage the development of such a
 82049 C-compatible one. These arguments did not sufficiently address the needs of current
 82050 application developers.

82051 2. Standardize the historical *dc*, possibly with minor modifications.
 82052 The consensus of the standard developers was that *dc* is a fundamentally less usable
 82053 language and that that would be far too severe a penalty for avoiding the issue of being
 82054 similar to but incompatible with C.

82055 3. Standardize the historical *bc*, possibly with minor modifications.
 82056 This was the approach taken. Most of the proponents of changing the language would not
 82057 have been satisfied until most or all of the incompatibilities with C were resolved. Since
 82058 most of the changes considered most desirable would break historical applications and
 82059 require significant modification to historical implementations, almost no modifications
 82060 were made. The one significant modification that was made was the replacement of the
 82061 historical *bc* assignment operators "=", and so on, with the more modern "+=", and so
 82062 on. The older versions are considered to be fundamentally flawed because of the lexical
 82063 ambiguity in uses like $a=-1$.

82064 In order to permit implementations to deal with backwards-compatibility as they see fit,
 82065 the behavior of this one ambiguous construct was made undefined. (At least three
 82066 implementations have been known to support this change already, so the degree of
 82067 change involved should not be great.)

82068 The '%' operator is the mathematical remainder operator when **scale** is zero. The behavior of
 82069 this operator for other values of **scale** is from historical implementations of *bc*, and has been
 82070 maintained for the sake of historical applications despite its non-intuitive nature.

82071 Historical implementations permit setting **ibase** and **obase** to a broader range of values. This
 82072 includes values less than 2, which were not seen as sufficiently useful to standardize. These
 82073 implementations do not interpret input properly for values of **ibase** that are greater than 16. This
 82074 is because numeric constants are recognized syntactically, rather than lexically, as described in
 82075 this volume of POSIX.1-2017. They are built from lexical tokens of single hexadecimal digits and
 82076 <period> characters. Since <blank> characters between tokens are not visible at the syntactic
 82077 level, it is not possible to recognize the multi-digit "digits" used in the higher bases properly.
 82078 The ability to recognize input in these bases was not considered useful enough to require
 82079 modifying these implementations. Note that the recognition of numeric constants at the
 82080 syntactic level is not a problem with conformance to this volume of POSIX.1-2017, as it does not
 82081 impact the behavior of conforming applications (and correct *bc* programs). Historical
 82082 implementations also accept input with all of the digits '0'-'9' and 'A'-'F' regardless of the
 82083 value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate,
 82084 the behavior when they appear is undefined, except for the common case of:

```
82085 ibase=8;
82086     /* Process in octal base. */
82087     ...
82088 ibase=A
82089     /* Restore decimal base. */
```

82090 In some historical implementations, if the expression to be written is an uninitialized array

82091 element, a leading <space> and/or up to four leading 0 characters may be output before the
82092 character zero. This behavior is considered a bug; it is unlikely that any currently conforming
82093 application relies on:

```
82094 echo 'b[3]' | bc
```

82095 returning 00000 rather than 0.

82096 Exact calculation of the number of fractional digits to output for a given value in a base other
82097 than 10 can be computationally expensive. Historical implementations use a faster
82098 approximation, and this is permitted. Note that the requirements apply only to values of **obase**
82099 that this volume of POSIX.1-2017 requires implementations to support (in particular, not to 1, 0,
82100 or negative bases, if an implementation supports them as an extension).

82101 Historical implementations of *bc* did not allow array parameters to be passed as the last
82102 parameter to a function. New implementations are encouraged to remove this restriction even
82103 though it is not required by the grammar.

82104 **FUTURE DIRECTIONS**

82105 None.

82106 **SEE ALSO**

82107 [Section 1.3](#) (on page 2335), *awk*

82108 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

82109 **CHANGE HISTORY**

82110 First released in Issue 4.

82111 **Issue 5**

82112 The FUTURE DIRECTIONS section is added.

82113 **Issue 6**

82114 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several
82115 interpretations of the ISO POSIX-2: 1993 standard.

82116 The normative text is reworded to avoid use of the term “must” for application requirements.

82117 **Issue 7**

82118 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82119 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0066 [584] and XCU/TC2-2008/0067
82120 [679] are applied.

82121 **NAME**82122 `bg` — run jobs in the background82123 **SYNOPSIS**82124 UP `bg [job_id...]`82125 **DESCRIPTION**

82126 If job control is enabled (see the description of `set -m`), the `bg` utility shall resume suspended jobs
 82127 from the current environment (see [Section 2.12](#), on page 2381) by running them as background
 82128 jobs. If the job specified by `job_id` is already a running background job, the `bg` utility shall have
 82129 no effect and shall exit successfully.

82130 Using `bg` to place a job into the background shall cause its process ID to become “known in the
 82131 current shell execution environment”, as if it had been started as an asynchronous list; see
 82132 [Section 2.9.3.1](#) (on page 2370).

82133 **OPTIONS**

82134 None.

82135 **OPERANDS**

82136 The following operand shall be supported:

82137 `job_id` Specify the job to be resumed as a background job. If no `job_id` operand is given,
 82138 the most recently suspended job shall be used. The format of `job_id` is described in
 82139 XBD [Section 3.204](#) (on page 66).

82140 **STDIN**

82141 Not used.

82142 **INPUT FILES**

82143 None.

82144 **ENVIRONMENT VARIABLES**82145 The following environment variables shall affect the execution of `bg`:

82146 `LANG` Provide a default value for the internationalization variables that are unset or null.
 82147 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 82148 variables used to determine the values of locale categories.)

82149 `LC_ALL` If set to a non-empty string value, override the values of all the other
 82150 internationalization variables.

82151 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
 82152 characters (for example, single-byte as opposed to multi-byte characters in
 82153 arguments).

82154 `LC_MESSAGES`

82155 Determine the locale that should be used to affect the format and contents of
 82156 diagnostic messages written to standard error.

82157 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.82158 **ASYNCHRONOUS EVENTS**

82159 Default.

82160 **STDOUT**82161 The output of `bg` shall consist of a line in the format:82162 `"[%d] %s\n", <job-number>, <command>`

82163 where the fields are as follows:

82164 <*job-number*> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using
82165 these utilities, the job can be identified by prefixing the job number with '% '.

82166 <*command*> The associated command that was given to the shell.

82167 **STDERR**

82168 The standard error shall be used only for diagnostic messages.

82169 **OUTPUT FILES**

82170 None.

82171 **EXTENDED DESCRIPTION**

82172 None.

82173 **EXIT STATUS**

82174 The following exit values shall be returned:

82175 0 Successful completion.

82176 >0 An error occurred.

82177 **CONSEQUENCES OF ERRORS**

82178 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the
82179 background.

82180 **APPLICATION USAGE**

82181 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see
82182 XBD [Chapter 11](#) (on page 199). At that point, *bg* can put the job into the background. This is
82183 most effective when the job is expecting no terminal input and its output has been redirected to
82184 non-terminal files. A background job can be forced to stop when it has terminal output by
82185 issuing the command:

```
82186 stty tostop
```

82187 A background job can be stopped with the command:

```
82188 kill -s stop job ID
```

82189 The *bg* utility does not work as expected when it is operating in its own utility execution
82190 environment because that environment has no suspended jobs. In the following examples:

```
82191 ... | xargs bg  
82192 (bg)
```

82193 each *bg* operates in a different environment and does not share its parent shell's understanding
82194 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

82195 **EXAMPLES**

82196 None.

82197 **RATIONALE**

82198 The extensions to the shell specified in this volume of POSIX.1-2017 have mostly been based on
82199 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
82200 based on the KornShell. The standard developers examined the characteristics of the C shell
82201 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
82202 the KornShell versions were selected for this volume of POSIX.1-2017 to maintain a degree of
82203 uniformity with the rest of the KornShell features selected (such as the very popular command
82204 line editing features).

82205 The *bg* utility is expected to wrap its output if the output exceeds the number of display
82206 columns.

82207 **FUTURE DIRECTIONS**

82208 None.

82209 **SEE ALSO**

82210 [Section 2.9.3.1](#) (on page 2370), *fg*, *kill*, *jobs*, *wait*

82211 XBD [Section 3.204](#) (on page 66), [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199)

82212 **CHANGE HISTORY**

82213 First released in Issue 4.

82214 **Issue 6**

82215 This utility is marked as part of the User Portability Utilities option.

82216 The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory
82217 in this version. This is a FIPS requirement.

82218 **Issue 7**

82219 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82220 **NAME**

82221 c99 ‡compile standard C programs

82222 **SYNOPSIS**

```
82223 CD c99 [options...] pathname [[pathname] [-I directory]
82224 [-L directory] [-l library]]...
```

82225 **DESCRIPTION**

82226 The *c99* utility is an interface to the standard C compilation system; it shall accept source code
 82227 conforming to the ISO C standard. The system conceptually consists of a compiler and link
 82228 editor. The input files referenced by *pathname* operands and *-I* option-arguments shall be
 82229 compiled and linked to produce an executable file. (It is unspecified whether the linking occurs
 82230 entirely within the operation of *c99*; some implementations may produce objects that are not
 82231 fully resolved until the file is executed.)

82232 If the *-c* option is specified, for all *pathname* operands of the form *file.c*, the files:

82233 $\$(\text{basename } \textit{pathname} \textit{.c}) \textit{.o}$

82234 shall be created as the result of successful compilation. If the *-c* option is not specified, it is
 82235 unspecified whether such *.o* files are created or deleted for the *file.c* operands.

82236 If there are no options that prevent link editing (such as *-c* or *-E*), and all input files compile and
 82237 link without error, the resulting executable file shall be written according to the *-o outfile* option
 82238 (if present) or to the file **a.out**.

82239 The executable file shall be created as specified in [Section 1.1.1.4](#) (on page 2328), except that the
 82240 file permission bits shall be set to:

82241 $S_IRWXO \mid S_IRWXG \mid S_IRWXU$

82242 and the bits specified by the *umask* of the process shall be cleared.

82243 **OPTIONS**

82244 The *c99* utility shall conform to XBD [Section 12.2](#) (on page 216), except that:

82245 Options can be interspersed with operands.

82246 The order of specifying the *-L* and *-I* options, and the order of specifying *-I* options with
 82247 respect to *pathname* operands is significant.

82248 Conforming applications shall specify each option separately; that is, grouping option
 82249 letters (for example, *-cO*) need not be recognized by all implementations.

82250 The following options shall be supported:

82251 *-c* Suppress the link-edit phase of the compilation, and do not remove any object files
 82252 that are produced.

82253 *-D name[=value]*

82254 Define *name* as if by a C-language **#define** directive. If no *=value* is given, a value of
 82255 1 shall be used. The *-D* option has lower precedence than the *-U* option. That is, if
 82256 *name* is used in both a *-U* and a *-D* option, *name* shall be undefined regardless of
 82257 the order of the options. Additional implementation-defined *names* may be
 82258 provided by the compiler. Implementations shall support at least 2 048 bytes of *-D*
 82259 definitions and 256 *names*.

- 82260 **-E** Copy C-language source files to standard output, executing all preprocessor directives; no compilation shall be performed. If any operand is not a text file, the effects are unspecified.
- 82261
- 82262
- 82263 **-g** Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.
- 82264
- 82265
- 82266 **-I *directory*** Change the algorithm for searching for headers whose names are not absolute pathnames to look in the *directory* named by the *directory* pathname before looking in the usual places. Thus, headers whose names are enclosed in double-quotes (" ") shall be searched for first in the directory of the file with the **#include** line, then in directories named in **-I** options, and last in the usual places. For headers whose names are enclosed in angle brackets ("**<>**"), the header shall be searched for only in directories named in **-I** options and then in the usual places. Directories named in **-I** options shall be searched in the order specified. If the **-I** option is used to specify a directory that is one of the usual places searched by default, the results are unspecified. Implementations shall support at least ten instances of this option in a single *c99* command invocation.
- 82267
- 82268
- 82269
- 82270
- 82271
- 82272
- 82273
- 82274
- 82275
- 82276
- 82277 **-L *directory*** Change the algorithm of searching for the libraries named in the **-I** objects to look in the directory named by the *directory* pathname before looking in the usual places. Directories named in **-L** options shall be searched in the order specified. If the **-L** option is used to specify a directory that is one of the usual places searched by default, the results are unspecified. Implementations shall support at least ten instances of this option in a single *c99* command invocation. If a directory specified by a **-L** option contains files with names starting with any of the strings "libc.", "libl.", "libpthread.", "libm.", "librt.", "libtrace.", "libxnet.", or "liby.", the results are unspecified.
- 82278
- 82279
- 82280
- 82281
- 82282
- 82283
- 82284
- 82285
- 82286 **-l *library*** Search the library named **liblibrary.a**. A library shall be searched when its name is encountered, so the placement of a **-l** option is significant. Several standard libraries can be specified in this manner, as described in the EXTENDED DESCRIPTION section. Implementations may recognize implementation-defined suffixes other than **.a** as denoting libraries.
- 82287
- 82288
- 82289
- 82290
- 82291 **-O *optlevel*** Specify the level of code optimization. If the *optlevel* option-argument is the digit '0', all special code optimizations shall be disabled. If it is the digit '1', the nature of the optimization is unspecified. If the **-O** option is omitted, the nature of the system's default optimization is unspecified. It is unspecified whether code generated in the presence of the **-O 0** option is the same as that generated when **-O** is omitted. Other *optlevel* values may be supported.
- 82292
- 82293
- 82294
- 82295
- 82296
- 82297 **-o *outfile*** Use the pathname *outfile*, instead of the default **a.out**, for the executable file produced. If the **-o** option is present with **-c** or **-E**, the result is unspecified.
- 82298
- 82299 **-s** Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the *exec* family defined in the System Interfaces volume of POSIX.1-2017 has been removed (stripped). If both **-g** and **-s** options are present, the action taken is unspecified.
- 82300
- 82301
- 82302
- 82303 **-U *name*** Remove any initial definition of *name*.
- 82304 Multiple instances of the **-D**, **-I**, **-L**, **-l**, and **-U** options can be specified.

82305 **OPERANDS**

82306 The application shall ensure that at least one *pathname* operand is specified. The following forms
82307 for *pathname* operands shall be supported:

82308 *file.c* A C-language source file to be compiled and optionally linked. The application
82309 shall ensure that the operand is of this form if the `-c` option is used.

82310 *file.a* A library of object files typically produced by the *ar* utility, and passed directly to
82311 the link editor. Implementations may recognize implementation-defined suffixes
82312 other than `.a` as denoting object file libraries.

82313 *file.o* An object file produced by *c99 -c* and passed directly to the link editor.
82314 Implementations may recognize implementation-defined suffixes other than `.o`
82315 as denoting object files.

82316 The processing of other files is implementation-defined.

82317 **STDIN**

82318 Not used.

82319 **INPUT FILES**

82320 Each input file shall be one of the following: a text file containing a C-language source program,
82321 an object file in the format produced by *c99 -c*, or a library of object files, in the format produced
82322 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
82323 that produce files in these formats. Additional input file formats are implementation-defined.

82324 **ENVIRONMENT VARIABLES**

82325 The following environment variables shall affect the execution of *c99*:

82326 *LANG* Provide a default value for the internationalization variables that are unset or null.
82327 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
82328 variables used to determine the values of locale categories.)

82329 *LC_ALL* If set to a non-empty string value, override the values of all the other
82330 internationalization variables.

82331 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
82332 characters (for example, single-byte as opposed to multi-byte characters in
82333 arguments and input files).

82334 *LC_MESSAGES*

82335 Determine the locale that should be used to affect the format and contents of
82336 diagnostic messages written to standard error.

82337 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

82338 *TMPDIR* Provide a pathname that should override the default directory for temporary files,
82339 XSI if any. On XSI-conforming systems, provide a pathname that shall override the
82340 default directory for temporary files, if any.

82341 **ASYNCHRONOUS EVENTS**

82342 Default.

82343 **STDOUT**

82344 If more than one *pathname* operand ending in `.c` (or possibly other unspecified suffixes) is given,
82345 for each such file:

82346 "%s:\n", *<pathname>*

82347 may be written. These messages, if written, shall precede the processing of each input file; they

82348 shall not be written to the standard output if they are written to the standard error, as described
82349 in the STDERR section.

82350 If the **-E** option is specified, the standard output shall be a text file that represents the results of
82351 the preprocessing stage of the language; it may contain extra information appropriate for
82352 subsequent compilation passes.

82353 **STDERR**

82354 The standard error shall be used only for diagnostic messages. If more than one *pathname*
82355 operand ending in **.c** (or possibly other unspecified suffixes) is given, for each such file:

82356 "%s:\n", *<pathname>*

82357 may be written to allow identification of the diagnostic and warning messages with the
82358 appropriate input file. These messages, if written, shall precede the processing of each input file;
82359 they shall not be written to the standard error if they are written to the standard output, as
82360 described in the STDOUT section.

82361 This utility may produce warning messages about certain conditions that do not warrant
82362 returning an error (non-zero) exit value.

82363 **OUTPUT FILES**

82364 Object files or executable files or both are produced in unspecified formats. If the pathname of
82365 an object file or executable file to be created by *c99* resolves to an existing directory entry for a
82366 file that is not a regular file, it is unspecified whether *c99* shall attempt to create the file or shall
82367 issue a diagnostic and exit with a non-zero exit status.

82368 **EXTENDED DESCRIPTION**

82369 **Standard Libraries**

82370 The *c99* utility shall recognize the following **-l** options for standard libraries:

82371 **-l c** This option shall make available all interfaces referenced in the System Interfaces
82372 volume of POSIX.1-2017, with the possible exception of those interfaces listed as
82373 residing in **<aio.h>**, **<arpa/inet.h>**, **<complex.h>**, **<fenv.h>**, **<math.h>**,
82374 **<mqueue.h>**, **<netdb.h>**, **<net/if.h>**, **<netinet/in.h>**, **<pthread.h>**, **<sched.h>**,
82375 **<semaphore.h>**, **<spawn.h>**, **<sys/socket.h>**, *pthread_kill()*, and *pthread_sigmask()*
82376 in **<signal.h>**, **<trace.h>**, interfaces marked as optional in **<sys/mman.h>**,
82377 interfaces marked as ADV (Advisory Information) in **<fcntl.h>**, and interfaces
82378 beginning with the prefix *clock_* or *timer_* in **<time.h>**. This option shall not be
82379 required to be present to cause a search of this library.

82380 **-l l** This option shall make available all interfaces required by the C-language output
82381 of *lex* that are not made available through the **-l c** option.

82382 **-l pthread** This option shall make available all interfaces referenced in **<pthread.h>** and
82383 *pthread_kill()* and *pthread_sigmask()* referenced in **<signal.h>**. An implementation
82384 may search this library in the absence of this option.

82385 **-l m** This option shall make available all interfaces referenced in **<math.h>**,
82386 **<complex.h>**, and **<fenv.h>**. An implementation may search this library in the
82387 absence of this option.

82388 **-l rt** This option shall make available all interfaces referenced in **<aio.h>**, **<mqueue.h>**,
82389 **<sched.h>**, **<semaphore.h>**, and **<spawn.h>**, interfaces marked as optional in
82390 **<sys/mman.h>**, interfaces marked as ADV (Advisory Information) in **<fcntl.h>**,
82391 and interfaces beginning with the prefix *clock_* and *timer_* in **<time.h>**. An

- 82392 implementation may search this library in the absence of this option.
- 82393 OB **-l trace** This option shall make available all interfaces referenced in `<trace.h>`. An
82394 implementation may search this library in the absence of this option.
- 82395 **-l xnet** This option shall make available all interfaces referenced in `<arpa/inet.h>`,
82396 `<netdb.h>`, `<net/if.h>`, `<netinet/in.h>`, and `<sys/socket.h>`. An implementation
82397 may search this library in the absence of this option.
- 82398 **-l y** This option shall make available all interfaces required by the C-language output
82399 of *yacc* that are not made available through the **-l c** option.
- 82400 In the absence of options that inhibit invocation of the link editor, such as **-c** or **-E**, the *c99* utility
82401 shall cause the equivalent of a **-l c** option to be passed to the link editor after the last *pathname*
82402 operand or **-l** option, causing it to be searched after all other object files and libraries are loaded.
- 82403 OB It is unspecified whether the libraries **libc.a**, **libl.a**, **libm.a**, **libpthread.a**, **librt.a**, **libtrace.a**,
82404 **libxnet.a**, or **liby.a** exist as regular files. The implementation may accept as **-l** option-arguments
82405 names of objects that do not exist as regular files.
- 82406 **External Symbols**
- 82407 The C compiler and link editor shall support the significance of external symbols up to a length
82408 of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-
82409 defined maximum symbol length is unspecified.
- 82410 The compiler and link editor shall support a minimum of 511 external symbols per source or
82411 object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be
82412 written to the standard output if the implementation-defined limit is exceeded; other actions are
82413 unspecified.
- 82414 **Header Search**
- 82415 If a file with the same name as one of the standard headers defined in XBD [Chapter 13](#) (on page
82416 219), not provided as part of the implementation, is placed in any of the usual places that are
82417 searched by default for headers, the results are unspecified.
- 82418 **Programming Environments**
- 82419 All implementations shall support one of the following programming environments as a default.
82420 Implementations may support more than one of the following programming environments.
82421 Applications can use *sysconf()* or *getconf* to determine which programming environments are
82422 supported.

82423

Table 4-4 Programming Environments: Type Sizes

82424

82425

82426

82427

82428

82429

Programming Environment <i>getconf</i> Name	Bits in int	Bits in long	Bits in pointer	Bits in off_t
_POSIX_V7_ILP32_OFF32	32	32	32	32
_POSIX_V7_ILP32_OFFBIG	32	32	32	≥64
_POSIX_V7_LP64_OFF64	32	64	64	64
_POSIX_V7_LP64_OFFBIG	≥32	≥64	≥64	≥64

82430

82431

All implementations shall support one or more environments where the widths of the following types are no greater than the width of type **long**:

82432

82433

82434

82435

82436

blksize_t	ptrdiff_t	tcflag_t
cc_t	size_t	wchar_t
mode_t	speed_t	wint_t
nfds_t	ssize_t	
pid_t	suseconds_t	

82437

82438

82439

82440

82441

82442

82443

82444

The executable files created when these environments are selected shall be in a proper format for execution by the *exec* family of functions. Each environment may be one of the ones in Table 4-4, or it may be another environment. The names for the environments that meet this requirement shall be output by a *getconf* command using the `POSIX_V7_WIDTH_RESTRICTED_ENVS` argument, as a <newline>-separated list of names suitable for use with the *getconf* `-v` option. If more than one environment meets the requirement, the names of all such environments shall be output on separate lines. Any of these names can then be used in a subsequent *getconf* command to obtain the flags specific to that environment with the following suffixes added as appropriate:

82445

`_CFLAGS` To get the C compiler flags.

82446

`_LDFLAGS` To get the linker/loader flags.

82447

`_LIBS` To get the libraries.

82448

This requirement may be removed in a future version.

82449

82450

82451

82452

82453

When this utility processes a file containing a function called *main()*, it shall be defined with a return type equivalent to **int**. Using return from the initial call to *main()* shall be equivalent (other than with respect to language scope issues) to calling *exit()* with the returned value. Reaching the end of the initial call to *main()* shall be equivalent to calling *exit(0)*. The implementation shall not declare a prototype for this function.

82454

82455

82456

82457

82458

82459

82460

82461

Implementations provide configuration strings for C compiler flags, linker/loader flags, and libraries for each supported environment. When an application needs to use a specific programming environment rather than the implementation default programming environment while compiling, the application shall first verify that the implementation supports the desired environment. If the desired programming environment is supported, the application shall then invoke *c99* with the appropriate C compiler flags as the first options for the compile, the appropriate linker/loader flags after any other options except `-I` but before any operands or `-l` options, and the appropriate libraries at the end of the operands and `-l` options.

82462

82463

82464

Conforming applications shall not attempt to link together object files compiled for different programming models. Applications shall also be aware that binary data placed in shared memory or in files might not be recognized by applications built for other programming models.

82465

Table 4-5 Programming Environments: *c99* Arguments

Programming Environment <i>getconf</i> Name	Use	<i>c99</i> Arguments <i>getconf</i> Name
_POSIX_V7_ILP32_OFF32	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_ILP32_OFF32_CFLAGS POSIX_V7_ILP32_OFF32_LDFLAGS POSIX_V7_ILP32_OFF32_LIBS
_POSIX_V7_ILP32_OFFBIG	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_ILP32_OFFBIG_CFLAGS POSIX_V7_ILP32_OFFBIG_LDFLAGS POSIX_V7_ILP32_OFFBIG_LIBS
_POSIX_V7_LP64_OFF64	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_LP64_OFF64_CFLAGS POSIX_V7_LP64_OFF64_LDFLAGS POSIX_V7_LP64_OFF64_LIBS
_POSIX_V7_LP64_OFFBIG	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_LP64_OFFBIG_CFLAGS POSIX_V7_LP64_OFFBIG_LDFLAGS POSIX_V7_LP64_OFFBIG_LIBS

82480

82481

82482

82483

In addition to the type size programming environments above, all implementations also support a multi-threaded programming environment that is orthogonal to all of the programming environments listed above. The *getconf* utility can be used to get flags for the threaded programming environment, as indicated in [Table 4-6](#).

82484

Table 4-6 Threaded Programming Environment: *c99* Arguments

Programming Environment <i>getconf</i> Name	Use	<i>c99</i> Arguments <i>getconf</i> Name
_POSIX_THREADS	C Compiler Flags Linker/Loader Flags	POSIX_V7_THREADS_CFLAGS POSIX_V7_THREADS_LDFLAGS

82489

82490

These programming environment flags may be used in conjunction with any of the type size programming environments supported by the implementation.

EXIT STATUS

82491

The following exit values shall be returned:

82492

0 Successful compilation or link edit.

82493

>0 An error occurred.

CONSEQUENCES OF ERRORS

82495

82496

82497

82498

82499

82500

82501

When *c99* encounters a compilation error that causes an object file not to be created, it shall write a diagnostic to standard error and continue to compile other source code operands, but it shall not perform the link phase and it shall return a non-zero exit status. If the link edit is unsuccessful, a diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A conforming application shall rely on the exit status of *c99*, rather than on the existence or mode of the executable file.

82502 **APPLICATION USAGE**

82503 Since the *c99* utility usually creates files in the current directory during the compilation process,
82504 it is typically necessary to run the *c99* utility in a directory in which a file can be created.

82505 On systems providing POSIX Conformance (see XBD [Chapter 2](#), on page 15), *c99* is required
82506 only with the C-Language Development option; XSI-conformant systems always provide *c99*.

82507 Some historical implementations have created **.o** files when **-c** is not specified and more than
82508 one source file is given. Since this area is left unspecified, the application cannot rely on **.o** files
82509 being created, but it also must be prepared for any related **.o** files that already exist being deleted
82510 at the completion of the link edit.

82511 There is the possible implication that if a user supplies versions of the standard functions (before
82512 they would be encountered by an implicit **-l c** or explicit **-l m**), that those versions would be
82513 used in place of the standard versions. There are various reasons this might not be true
82514 (functions defined as macros, manipulations for clean name space, and so on), so the existence of
82515 files named in the same manner as the standard libraries within the **-L** directories is explicitly
82516 stated to produce unspecified behavior.

82517 All of the functions specified in the System Interfaces volume of POSIX.1-2017 may be made
82518 visible by implementations when the Standard C Library is searched. Conforming applications
82519 must explicitly request searching the other standard libraries when functions made visible by
82520 those libraries are used.

82521 In the ISO C standard the mapping from physical source characters to the C source character set
82522 is implementation-defined. Implementations may strip white-space characters before the
82523 terminating **<newline>** of a (physical) line as part of this mapping and, as a consequence of this,
82524 one or more white-space characters (and no other characters) between a **<backslash>** character
82525 and the **<newline>** character that terminates the line produces implementation-defined results.
82526 Portable applications should not use such constructs.

82527 Some *c99* compilers not conforming to POSIX.1-2017 do not support trigraphs by default.

82528 **EXAMPLES**

82529 1. The following usage example compiles **foo.c** and creates the executable file **foo**:

```
82530 c99 -o foo foo.c
```

82531 The following usage example compiles **foo.c** and creates the object file **foo.o**:

```
82532 c99 -c foo.c
```

82533 The following usage example compiles **foo.c** and creates the executable file **a.out**:

```
82534 c99 foo.c
```

82535 The following usage example compiles **foo.c**, links it with **bar.o**, and creates the
82536 executable file **a.out**. It may also create and leave **foo.o**:

```
82537 c99 foo.c bar.o
```

82538 2. The following example shows how an application using threads interfaces can test for
82539 support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer**
82540 types and an **off_t** type using at least 64 bits:

```
82541 offbig_env=$(getconf _POSIX_V7_ILP32_OFFBIG)
82542 if [ $offbig_env != "-1" ] && [ $offbig_env != "undefined" ]
82543 then
82544     c99 $(getconf POSIX_V7_ILP32_OFFBIG_CFLAGS) \
```

```

82545     $(getconf POSIX_V7_THREADS_CFLAGS) -D_XOPEN_SOURCE=700 \
82546     $(getconf POSIX_V7_ILP32_OFFBIG_LDFLAGS) \
82547     $(getconf POSIX_V7_THREADS_LDFLAGS) foo.c -o foo \
82548     $(getconf POSIX_V7_ILP32_OFFBIG_LIBS) \
82549     -l pthread
82550 else
82551     echo ILP32_OFFBIG programming environment not supported
82552     exit 1
82553 fi

```

3. The following examples clarify the use and interactions of `-L` and `-I` options.

Consider the case in which module `a.c` calls function `f()` in library `libQ.a`, and module `b.c` calls function `g()` in library `libp.a`. Assume that both libraries reside in `/a/b/c`. The command line to compile and link in the desired way is:

```
c99 -L /a/b/c main.o a.c -l Q b.c -l p
```

In this case the `-L` option need only precede the first `-I` option, since both `libQ.a` and `libp.a` reside in the same directory.

Multiple `-L` options can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new `libp.a`, in `/a/a/a`, but still wants `f()` from `/a/b/c/libQ.a`:

```
c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p
```

In this example, the linker searches the `-L` options in the order specified, and finds `/a/a/a/libp.a` before `/a/b/c/libp.a` when resolving references for `b.c`. The order of the `-I` options is still important, however.

4. The following example shows how an application can use a programming environment where the widths of the following types:

blksize_t, cc_t, mode_t, nfsd_t, pid_t, ptrdiff_t, size_t, speed_t, ssize_t, suseconds_t, tflag_t, wchar_t, wint_t

are no greater than the width of type `long`:

```

82572 # First choose one of the listed environments ...
82573
82574 # ... if there are no additional constraints, the first one will do:
82575 CENV=$(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS | head -n 1)
82576
82577 # ... or, if an environment that supports large files is preferred,
82578 # look for names that contain "OFF64" or "OFFBIG". (This chooses
82579 # the last one in the list if none match.)
82580 for CENV in $(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS)
82581 do
82582     case $CENV in
82583         *OFF64*|*OFFBIG*) break ;;
82584     esac
82585 done
82586
82587 # The chosen environment name can now be used like this:
82588
82589 c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200809L \
82590 $(getconf ${CENV}_LDFLAGS) foo.c -o foo \
82591 $(getconf ${CENV}_LIBS)

```

82589 **RATIONALE**

82590 The *c99* utility is based on the *c89* utility originally introduced in the ISO POSIX-2:1993
82591 standard.

82592 Some of the changes from *c89* include the ability to intersperse options and operands (which
82593 many *c89* implementations allowed despite it not being specified), the description of `-l` as an
82594 option instead of an operand, and the modification to the contents of the Standard Libraries
82595 section to account for new headers and options; for example, `<spawn.h>` added to the
82596 description of `-l rt`, and `-l trace` added for the Tracing option.

82597 POSIX.1-2017 specifies that the *c99* utility must be able to use regular files for `*.o` files and for
82598 `a.out` files. Implementations are free to overwrite existing files of other types when attempting to
82599 create object files and executable files, but are not required to do so. If something other than a
82600 regular file is specified and using it fails for any reason, *c99* is required to issue a diagnostic
82601 message and exit with a non-zero exit status. But for some file types, the problem may not be
82602 noticed for a long time. For example, if a FIFO named `a.out` exists in the current directory, *c99*
82603 may attempt to open `a.out` and will hang in the `open()` call until another process opens the FIFO
82604 for reading. Then *c99* may write most of the `a.out` to the FIFO and fail when it tries to seek back
82605 close to the start of the file to insert a timestamp (FIFOs are not seekable files). The *c99* utility is
82606 also allowed to issue a diagnostic immediately if it encounters an `a.out` or `*.o` file that is not a
82607 regular file. For portable use, applications should ensure that any `a.out`, `-o` option-argument, or
82608 `*.o` files corresponding to any `*.c` files do not conflict with names already in use that are not
82609 regular files or symbolic links that point to regular files.

82610 On many systems, multi-threaded applications run in a programming environment that is
82611 distinct from that used by single-threaded applications. This multi-threaded programming
82612 environment (in addition to needing to specify `-l pthread` at link time) may require additional
82613 flags to be set when headers are processed at compile time (`-D_REENTRANT` being common).
82614 This programming environment is orthogonal to the type size programming environments
82615 discussed above and listed in Table 4-4 (on page 2547). This version of the standard adds *getconf*
82616 utility calls to provide the C compiler flags and linker/loader flags needed to support multi-
82617 threaded applications. Note that on a system where single-threaded applications are a special
82618 case of a multi-threaded application, both of these *getconf* calls may return NULL strings; on
82619 other implementations both of these strings may be non-NULL strings.

82620 The C standardization committee invented trigraphs (e.g., "??!" to represent '|') to address
82621 character portability problems in development environments based on national variants of the
82622 7-bit ISO/IEC 646:1991 standard character set. However, these environments were already
82623 obsolete by the time the first ISO C standard was published, and in practice trigraphs have not
82624 been used for their intended purpose, and usually are intended to have their original meaning in
82625 K&R C. For example, in practice a C-language source string like "What??!" is usually intended
82626 to end in two <question-mark> characters and an <exclamation-mark>, not in '| '.

82627 When the `-E` option is used, execution of some `#pragma` preprocessor directives may simply
82628 result in a copy of the directive being included in the output as part of the allowed extra
82629 information used by subsequent compilation passes (see `STDOUT`).

82630 **FUTURE DIRECTIONS**

82631 Unlike all of the other non-OB-shaded utilities in this standard, a utility by this name probably
82632 will not appear in the next version of this standard. This utility's name is tied to the current
82633 revision of the ISO C standard at the time this standard is approved. Since the ISO C standard
82634 and this standard are maintained by different organizations on different schedules, we cannot
82635 predict what the compiler will be named in the next version of the standard.

82636 **SEE ALSO**

- 82637 [Section 1.1.1.4](#) (on page 2328), [ar](#), [getconf](#), [make](#), [nm](#), [strip](#), [umask](#)
- 82638 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216), [Chapter 13](#) (on page 219)
- 82639 [XSH exec](#), [sysconf\(\)](#)

82640 **CHANGE HISTORY**

- 82641 First released in Issue 6. Included for alignment with the ISO/IEC 9899: 1999 standard.
- 82642 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/12 is applied, correcting the EXTENDED DESCRIPTION of `-l c` and `-l m`. Previously, the text did not take into account the presence of the `c99` math headers.
- 82643
- 82644
- 82645 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/13 is applied, changing the reference to the `libxnet` library to `libxnet.a`.
- 82646
- 82647 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/5 is applied, updating the OPTIONS section, so that the names of files contained in the directory specified by the `-L` option are not assumed to end in the `.a` suffix. The set of library prefixes is also updated.
- 82648
- 82649
- 82650 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/6 is applied, removing the lead underscore from the `POSIX_V6_WIDTH_RESTRICTED_ENVS` variable in the EXTENDED DESCRIPTION and the EXAMPLES sections.
- 82651
- 82652
- 82653 **Issue 7**
- 82654 Austin Group Interpretation 1003.1-2001 #020 (SD5-XCU-ERN-10) is applied, adding to the OUTPUT FILES section and also adding associated RATIONALE.
- 82655
- 82656 Austin Group Interpretation 1003.1-2001 #095 is applied, clarifying the `-l library` operand.
- 82657 Austin Group Interpretation 1003.1-2001 #166 is applied.
- 82658 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the handling of trailing white-space characters.
- 82659
- 82660 Austin Group Interpretation 1003.1-2001 #191 is applied, adding APPLICATION USAGE and RATIONALE regarding C-language trigraphs.
- 82661
- 82662 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply (options can be interspersed with operands).
- 82663
- 82664 SD5-XCU-ERN-11 is applied, adding the `<net/if.h>` header to the descriptions of `-l c` and `-l xnet`.
- 82665
- 82666 SD5-XCU-ERN-65 is applied, updating the EXAMPLES section.
- 82667 SD5-XCU-ERN-67 and SD5-XCU-ERN-97 are applied, updating the SYNOPSIS.
- 82668 SD5-XCU-ERN-133 is applied, updating the EXTENDED DESCRIPTION.
- 82669 The `getconf` variables for the supported programming environments are updated to be V7.
- 82670 The `-l trace` operand is marked obsolescent.
- 82671 The `c99` reference page is rewritten to describe `-l` as an option rather than an operand.
- 82672 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0068 [129], XCU/TC1-2008/0069 [187], XCU/TC1-2008/0070 [187], XCU/TC1-2008/0071 [131], XCU/TC1-2008/0072 [187], and XCU/TC1-2008/0073 [364,430] are applied.
- 82673
- 82674

82675
82676
82677

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0068 [650], XCU/TC2-2008/0069 [670], XCU/TC2-2008/0070 [638], XCU/TC2-2008/0071 [650], and XCU/TC2-2008/0072 [784] are applied.

82678 **NAME**

82679 cal ‡print a calendar

82680 **SYNOPSIS**82681 XSI cal `[[month] year]`82682 **DESCRIPTION**

82683 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from
 82684 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,
 82685 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on
 82686 September 14, 1752.

82687 If no operands are given, *cal* shall produce a one-month calendar for the current month in the
 82688 current year. If only the *year* operand is given, *cal* shall produce a calendar for all twelve months
 82689 in the given calendar year. If both *month* and *year* operands are given, *cal* shall produce a one-
 82690 month calendar for the given month in the given year.

82691 **OPTIONS**

82692 None.

82693 **OPERANDS**

82694 The following operands shall be supported:

82695 *month* Specify the month to be displayed, represented as a decimal integer from 1
 82696 (January) to 12 (December).

82697 *year* Specify the year for which the calendar is displayed, represented as a decimal
 82698 integer from 1 to 9999.

82699 **STDIN**

82700 Not used.

82701 **INPUT FILES**

82702 None.

82703 **ENVIRONMENT VARIABLES**82704 The following environment variables shall affect the execution of *cal*:

82705 *LANG* Provide a default value for the internationalization variables that are unset or null.
 82706 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 82707 variables used to determine the values of locale categories.)

82708 *LC_ALL* If set to a non-empty string value, override the values of all the other
 82709 internationalization variables.

82710 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 82711 characters (for example, single-byte as opposed to multi-byte characters in
 82712 arguments).

82713 *LC_MESSAGES*

82714 Determine the locale that should be used to affect the format and contents of
 82715 diagnostic messages written to standard error, and informative messages written
 82716 to standard output.

82717 *LC_TIME* Determine the format and contents of the calendar.

82718 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

- 82719 *TZ* Determine the timezone used to calculate the value of the current month.
- 82720 **ASYNCHRONOUS EVENTS**
- 82721 Default.
- 82722 **STDOUT**
- 82723 The standard output shall be used to display the calendar, in an unspecified format.
- 82724 **STDERR**
- 82725 The standard error shall be used only for diagnostic messages.
- 82726 **OUTPUT FILES**
- 82727 None.
- 82728 **EXTENDED DESCRIPTION**
- 82729 None.
- 82730 **EXIT STATUS**
- 82731 The following exit values shall be returned:
- 82732 0 Successful completion.
- 82733 >0 An error occurred.
- 82734 **CONSEQUENCES OF ERRORS**
- 82735 Default.
- 82736 **APPLICATION USAGE**
- 82737 Note that:
- 82738 `cal 83`
- 82739 refers to A.D. 83, not 1983.
- 82740 **EXAMPLES**
- 82741 None.
- 82742 **RATIONALE**
- 82743 Earlier versions of this standard incorrectly required that the command:
- 82744 `cal 2000`
- 82745 write a one-month calendar for the current calendar month (no matter what the current year is) in the year 2000 to standard output. This did not match historic practice in any known version of the *cal* utility. The description has been updated to match historic practice. When only the *year* operand is given, *cal* writes a twelve-month calendar for the specified year.
- 82749 **FUTURE DIRECTIONS**
- 82750 A future version of this standard may support locale-specific recognition of the date of adoption of the Gregorian calendar.
- 82751 of the Gregorian calendar.
- 82752 **SEE ALSO**
- 82753 XBD [Chapter 8](#) (on page 173)
- 82754 **CHANGE HISTORY**
- 82755 First released in Issue 2.
- 82756 **Issue 6**
- 82757 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of the Gregorian calendar.
- 82758 the Gregorian calendar.

82759 **Issue 7**

82760 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82761 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0074 [56] and XCU/TC1-2008/0075
82762 [56] are applied.

82763 **NAME**82764 `cat` ‡concatenate and print files82765 **SYNOPSIS**82766 `cat [-u] [file...]`82767 **DESCRIPTION**82768 The *cat* utility shall read files in sequence and shall write their contents to the standard output in
82769 the same sequence.82770 **OPTIONS**82771 The *cat* utility shall conform to XBD [Section 12.2](#) (on page 216).

82772 The following option shall be supported:

82773 **-u** Write bytes from the input file to the standard output without delay as each is
82774 read.82775 **OPERANDS**

82776 The following operand shall be supported:

82777 *file* A pathname of an input file. If no *file* operands are specified, the standard input
82778 shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at
82779 that point in the sequence. The *cat* utility shall not close and reopen standard input
82780 when it is referenced in this way, but shall accept multiple occurrences of '-' as a
82781 *file* operand.82782 **STDIN**82783 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
82784 See the INPUT FILES section.82785 **INPUT FILES**

82786 The input files can be any file type.

82787 **ENVIRONMENT VARIABLES**82788 The following environment variables shall affect the execution of *cat*:82789 **LANG** Provide a default value for the internationalization variables that are unset or null.
82790 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
82791 variables used to determine the values of locale categories.)82792 **LC_ALL** If set to a non-empty string value, override the values of all the other
82793 internationalization variables.82794 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
82795 characters (for example, single-byte as opposed to multi-byte characters in
82796 arguments).82797 **LC_MESSAGES**82798 Determine the locale that should be used to affect the format and contents of
82799 diagnostic messages written to standard error.82800 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.82801 **ASYNCHRONOUS EVENTS**

82802 Default.

82803 STDOUT

82804 The standard output shall contain the sequence of bytes read from the input files. Nothing else
82805 shall be written to the standard output. If the standard output is a regular file, and is the same
82806 file as any of the input file operands, the implementation may treat this as an error.

82807 STDERR

82808 The standard error shall be used only for diagnostic messages.

82809 OUTPUT FILES

82810 None.

82811 EXTENDED DESCRIPTION

82812 None.

82813 EXIT STATUS

82814 The following exit values shall be returned:

82815 0 All input files were output successfully.

82816 >0 An error occurred.

82817 CONSEQUENCES OF ERRORS

82818 Default.

82819 APPLICATION USAGE

82820 The `-u` option has value in prototyping non-blocking reads from FIFOs. The intent is to support
82821 the following sequence:

```
82822 mkfifo foo  
82823 cat -u foo > /dev/tty13 &  
82824 cat -u > foo
```

82825 It is unspecified whether standard output is or is not buffered in the default case. This is
82826 sometimes of interest when standard output is associated with a terminal, since buffering may
82827 delay the output. The presence of the `-u` option guarantees that unbuffered I/O is available. It is
82828 implementation-defined whether the `cat` utility buffers output if the `-u` option is not specified.
82829 Traditionally, the `-u` option is implemented using the equivalent of the `setvbuf()` function
82830 defined in the System Interfaces volume of POSIX.1-2017.

82831 EXAMPLES

82832 The following command:

```
82833 cat myfile
```

82834 writes the contents of the file **myfile** to standard output.

82835 The following command:

```
82836 cat doc1 doc2 > doc.all
```

82837 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

82838 Because of the shell language mechanism used to perform output redirection, a command such
82839 as this:

```
82840 cat doc doc.end > doc
```

82841 causes the original data in **doc** to be lost before `cat` even begins execution. This is true whether
82842 the `cat` command fails with an error or silently succeeds (the specification allows both
82843 behaviors). In order to append the contents of **doc.end** without losing the original contents of
82844 **doc**, this command should be used instead:

82845 `cat doc.end >> doc`

82846 The command:

82847 `cat start - middle - end > file`

82848 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a
82849 single invocation of *cat*. Note, however, that if standard input is a regular file, this would be
82850 equivalent to the command:

82851 `cat start - middle /dev/null end > file`

82852 because the entire contents of the file would be consumed by *cat* the first time '-' was used as a
82853 *file* operand and an end-of-file condition would be detected immediately when '-' was
82854 referenced the second time.

82855 RATIONALE

82856 Historical versions of the *cat* utility include the `-e`, `-t`, and `-v`, options which permit the ends of
82857 lines, `<tab>` characters, and invisible characters, respectively, to be rendered visible in the
82858 output. The standard developers omitted these options because they provide too fine a degree of
82859 control over what is made visible, and similar output can be obtained using a command such as:

82860 `sed -n 1 pathname`

82861 The latter also has the advantage that its output is unambiguous, whereas the output of
82862 historical *cat -etv* is not.

82863 The `-s` option was omitted because it corresponds to different functions in BSD and System
82864 V-based systems. The BSD `-s` option to squeeze blank lines can be accomplished by the shell
82865 script shown in the following example:

```
82866 sed -n '  
82867 # Write non-empty lines.  
82868 ./ {  
82869     p  
82870     d  
82871 }  
82872 # Write a single empty line, then look for more empty lines.  
82873 /^$/ p  
82874 # Get next line, discard the held <newline> (empty line),  
82875 # and look for more empty lines.  
82876 :Empty  
82877 /^$/ {  
82878     N  
82879     s/././  
82880     b Empty  
82881 }  
82882 # Write the non-empty line before going back to search  
82883 # for the first in a set of empty lines.  
82884     p  
82885 '
```

82886 The System V `-s` option to silence error messages can be accomplished by redirecting the
82887 standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the
82888 same as the POSIX "empty line": a line consisting only of a `<newline>`.

82889 The BSD `-n` option was omitted because similar functionality can be obtained from the `-n`
82890 option of the *pr* utility.

82891 **FUTURE DIRECTIONS**

82892 None.

82893 **SEE ALSO**82894 *more*82895 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)82896 XSH *setvbuf()*82897 **CHANGE HISTORY**

82898 First released in Issue 2.

82899 **Issue 7**

82900 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

82901 SD5-XCU-ERN-174 is applied, changing the RATIONALE.

82902 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0073 [876] is applied.

82903 **NAME**82904 `cd` — change the working directory82905 **SYNOPSIS**82906 `cd [-L|-P] [directory]`82907 `cd -`82908 **DESCRIPTION**

82909 The `cd` utility shall change the working directory of the current shell execution environment (see
 82910 [Section 2.12](#), on page 2381) by executing the following steps in sequence. (In the following steps,
 82911 the symbol **curpath** represents an intermediate value used to simplify the description of the
 82912 algorithm used by `cd`. There is no requirement that **curpath** be made visible to the application.)

- 82913 1. If no *directory* operand is given and the *HOME* environment variable is empty or
 82914 undefined, the default behavior is implementation-defined and no further steps shall be
 82915 taken.
- 82916 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty
 82917 value, the `cd` utility shall behave as if the directory named in the *HOME* environment
 82918 variable was specified as the *directory* operand.
- 82919 3. If the *directory* operand begins with a <slash> character, set **curpath** to the operand and
 82920 proceed to step 7.
- 82921 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 82922 5. Starting with the first pathname in the <colon>-separated pathnames of *CDPATH* (see the
 82923 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the
 82924 concatenation of that pathname, a <slash> character if that pathname did not end with a
 82925 <slash> character, and the *directory* operand names a directory. If the pathname is null,
 82926 test if the concatenation of dot, a <slash> character, and the operand names a directory. In
 82927 either case, if the resulting string names an existing directory, set **curpath** to that string
 82928 and proceed to step 7. Otherwise, repeat this step with the next pathname in *CDPATH*
 82929 until all pathnames have been tested.
- 82930 6. Set **curpath** to the *directory* operand.
- 82931 7. If the `-P` option is in effect, proceed to step 10. If **curpath** does not begin with a <slash>
 82932 character, set **curpath** to the string formed by the concatenation of the value of *PWD*, a
 82933 <slash> character if the value of *PWD* did not end with a <slash> character, and **curpath**.
- 82934 8. The **curpath** value shall then be converted to canonical form as follows, considering each
 82935 component from beginning to end, in sequence:
 - 82936 a. Dot components and any <slash> characters that separate them from the next
 82937 component shall be deleted.
 - 82938 b. For each dot-dot component, if there is a preceding component and it is neither
 82939 root nor dot-dot, then:
 - 82940 i. If the preceding component does not refer (in the context of pathname
 82941 resolution with symbolic links followed) to a directory, then the `cd` utility
 82942 shall display an appropriate error message and no further steps shall be
 82943 taken.
 - 82944 ii. The preceding component, all <slash> characters separating the preceding
 82945 component from dot-dot, dot-dot, and all <slash> characters separating dot-
 82946 dot from the following component (if any) shall be deleted.

- 82947 c. An implementation may further simplify **curpath** by removing any trailing
 82948 <slash> characters that are not also leading <slash> characters, replacing multiple
 82949 non-leading consecutive <slash> characters with a single <slash>, and replacing
 82950 three or more leading <slash> characters with a single <slash>. If, as a result of
 82951 this canonicalization, the **curpath** variable is null, no further steps shall be taken.
- 82952 9. If **curpath** is longer than {PATH_MAX} bytes (including the terminating null) and the
 82953 *directory* operand was not longer than {PATH_MAX} bytes (including the terminating
 82954 null), then **curpath** shall be converted from an absolute pathname to an equivalent
 82955 relative pathname if possible. This conversion shall always be considered possible if the
 82956 value of *PWD*, with a trailing <slash> added if it does not already have one, is an initial
 82957 substring of **curpath**. Whether or not it is considered possible under other circumstances
 82958 is unspecified. Implementations may also apply this conversion if **curpath** is not longer
 82959 than {PATH_MAX} bytes or the *directory* operand was longer than {PATH_MAX} bytes.
- 82960 10. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with
 82961 **curpath** as the *path* argument. If these actions fail for any reason, the *cd* utility shall
 82962 display an appropriate error message and the remainder of this step shall not be
 82963 executed. If the **-P** option is not in effect, the *PWD* environment variable shall be set to
 82964 the value that **curpath** had on entry to step 9 (i.e., before conversion to a relative
 82965 pathname). If the **-P** option is in effect, the *PWD* environment variable shall be set to the
 82966 string that would be output by *pwd -P*. If there is insufficient permission on the new
 82967 directory, or on any parent of that directory, to determine the current working directory,
 82968 the value of the *PWD* environment variable is unspecified.

82969 If, during the execution of the above steps, the *PWD* environment variable is set, the *OLDPWD*
 82970 environment variable shall also be set to the value of the old working directory (that is the
 82971 current working directory immediately prior to the call to *cd*).

82972 OPTIONS

82973 The *cd* utility shall conform to XBD [Section 12.2](#) (on page 216).

82974 The following options shall be supported by the implementation:

- 82975 **-L** Handle the operand dot-dot logically; symbolic link components shall not be
 82976 resolved before dot-dot components are processed (see steps 8. and 9. in the
 82977 DESCRIPTION).
- 82978 **-P** Handle the operand dot-dot physically; symbolic link components shall be
 82979 resolved before dot-dot components are processed (see step 7. in the
 82980 DESCRIPTION).

82981 If both **-L** and **-P** options are specified, the last of these options shall be used and all others
 82982 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the
 82983 DESCRIPTION.

82984 OPERANDS

82985 The following operands shall be supported:

- 82986 *directory* An absolute or relative pathname of the directory that shall become the new
 82987 working directory. The interpretation of a relative pathname by *cd* depends on the
 82988 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an
 82989 empty string, the results are unspecified.
- 82990 **-** When a <hyphen-minus> is used as the operand, this shall be equivalent to the
 82991 command:
 82992

```
cd "$OLDPWD" && pwd
```

- 82993 which changes to the previous working directory and then writes its name.
- 82994 **STDIN**
- 82995 Not used.
- 82996 **INPUT FILES**
- 82997 None.
- 82998 **ENVIRONMENT VARIABLES**
- 82999 The following environment variables shall affect the execution of *cd*:
- 83000 *CDPATH* A <colon>-separated list of pathnames that refer to directories. The *cd* utility shall
- 83001 use this list in its attempt to change the directory, as described in the
- 83002 DESCRIPTION. An empty string in place of a directory pathname represents the
- 83003 current directory. If *CDPATH* is not set, it shall be treated as if it were an empty
- 83004 string.
- 83005 *HOME* The name of the directory, used when no *directory* operand is specified.
- 83006 *LANG* Provide a default value for the internationalization variables that are unset or null.
- 83007 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
- 83008 variables used to determine the values of locale categories.)
- 83009 *LC_ALL* If set to a non-empty string value, override the values of all the other
- 83010 internationalization variables.
- 83011 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 83012 characters (for example, single-byte as opposed to multi-byte characters in
- 83013 arguments).
- 83014 *LC_MESSAGES*
- 83015 Determine the locale that should be used to affect the format and contents of
- 83016 diagnostic messages written to standard error.
- 83017 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 83018 *OLDPWD* A pathname of the previous working directory, used by *cd -*.
- 83019 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets
- 83020 or unsets the value of *PWD*, the behavior of *cd* is unspecified.
- 83021 **ASYNCHRONOUS EVENTS**
- 83022 Default.
- 83023 **STDOUT**
- 83024 If a non-empty directory name from *CDPATH* is used, or if *cd -* is used, an absolute pathname of
- 83025 the new working directory shall be written to the standard output as follows:
- 83026 "%s\n", <*new directory*>
- 83027 Otherwise, there shall be no output.
- 83028 **STDERR**
- 83029 The standard error shall be used only for diagnostic messages.
- 83030 **OUTPUT FILES**
- 83031 None.

83032 **EXTENDED DESCRIPTION**

83033 None.

83034 **EXIT STATUS**

83035 The following exit values shall be returned:

83036 0 The directory was successfully changed.

83037 >0 An error occurred.

83038 **CONSEQUENCES OF ERRORS**

83039 The working directory shall remain unchanged.

83040 **APPLICATION USAGE**83041 Since *cd* affects the current shell execution environment, it is always provided as a shell regular
83042 built-in. If it is called in a subshell or separate utility execution environment, such as one of the
83043 following:83044 (cd /tmp)
83045 nohup cd
83046 find . -exec cd {} \;

83047 it does not affect the working directory of the caller's environment.

83048 The user must have execute (search) permission in *directory* in order to change to it.83049 **EXAMPLES**83050 The following template can be used to perform processing in the directory specified by *location*
83051 and end up in the current working directory in use before the first *cd* command was issued:83052 cd *location*
83053 if [\$? -ne 0]
83054 then
83055 print error message
83056 exit 1
83057 fi
83058 ... do whatever is desired as long as the OLDPWD environment variable
83059 is not modified
83060 cd -83061 **RATIONALE**83062 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of
83063 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.83064 A common extension when *HOME* is undefined is to get the login directory from the user
83065 database for the invoking user. This does not occur on System V implementations.83066 Some historical shells, such as the KornShell, took special actions when the directory name
83067 contained a dot-dot component, selecting the logical parent of the directory, rather than the
83068 actual parent directory; that is, it moved up one level toward the '/' in the pathname,
83069 remembering what the user typed, rather than performing the equivalent of:83070 `chdir("../");`83071 In such a shell, the following commands would not necessarily produce equivalent output for all
83072 directories:83073 `cd .. && ls ls ..`

83074 This behavior is now the default. It is not consistent with the definition of dot-dot in most

83075 historical practice; that is, while this behavior has been optionally available in the KornShell,
83076 other shells have historically not supported this functionality. The logical pathname is stored in
83077 the *PWD* environment variable when the *cd* utility completes and this value is used to construct
83078 the next directory name if *cd* is invoked with the *-L* option.

83079 **FUTURE DIRECTIONS**

83080 None.

83081 **SEE ALSO**

83082 [Section 2.12](#) (on page 2381), *pwd*

83083 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

83084 XSH *chdir()*

83085 **CHANGE HISTORY**

83086 First released in Issue 2.

83087 **Issue 6**

83088 The following new requirements on POSIX implementations derive from alignment with the
83089 Single UNIX Specification:

83090 The *cd -* operand, *PWD*, and *OLDPWD* are added.

83091 The *-L* and *-P* options are added to align with the IEEE P1003.2b draft standard. This also
83092 includes the introduction of a new description to include the effect of these options.

83093 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to
83094 make it clear that the *-L* and *-P* options are mutually-exclusive.

83095 **Issue 7**

83096 Austin Group Interpretation 1003.1-2001 #037 is applied.

83097 Austin Group Interpretation 1003.1-2001 #199 is applied, clarifying how the *cd* utility handles
83098 concatenation of two pathnames when the first pathname ends in a <slash> character.

83099 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83100 Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of “the operand”.

83101 Changes to the *pwd* utility and *PWD* environment variable have been made to match the
83102 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.

83103 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0076 [230], XCU/TC1-2008/0077
83104 [240], XCU/TC1-2008/0078 [240], and XCU/TC1-2008/0079 [123] are applied.

83105 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0074 [584] is applied.

83106 **NAME**

83107 cflow ‡generate a C-language flowgraph(DEVELOPMENT)

83108 **SYNOPSIS**

```
83109 XSI cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]...
83110 [-U dir]... file...
```

83111 **DESCRIPTION**

83112 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc*
 83113 source files, and attempt to build a graph, written to standard output, charting the external
 83114 references.

83115 **OPTIONS**

83116 The *cflow* utility shall conform to XBD Section 12.2 (on page 216), except that the order of the **-D**,
 83117 **-I**, and **-U** options (which are identical to their interpretation by *c99*) is significant.

83118 The following options shall be supported:

- 83119 **-d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure
 83120 that the argument *num* is a decimal integer. By default this is a very large number
 83121 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive
 83122 integer shall be ignored.
- 83123 **-i incl** Increase the number of included symbols. The *incl* option-argument is one of the
 83124 following characters:
- 83125 *x* Include external and static data symbols. The default shall be to include only
 83126 functions in the flowgraph.
- 83127 *_* (Underscore) Include names that begin with an <underscore>. The default
 83128 shall be to exclude these functions (and data if **-i x** is used).
- 83129 **-r** Reverse the caller: callee relationship, producing an inverted listing showing the
 83130 callers of each function. The listing shall also be sorted in lexicographical order by
 83131 callee.

83132 **OPERANDS**

83133 The following operand is supported:

- 83134 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by
 83135 **.I** shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c99* input, and **.i** as the
 83136 output of *c99* **-E**. Such files shall be processed as appropriate, determined by their
 83137 suffix.
- 83138 Files suffixed by **.s** (conventionally assembler source) may have more limited
 83139 information extracted from them.

83140 **STDIN**

83141 Not used.

83142 **INPUT FILES**

83143 The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

83144 **ENVIRONMENT VARIABLES**

83145 The following environment variables shall affect the execution of *cflow*:

- 83146 **LANG** Provide a default value for the internationalization variables that are unset or null.
 83147 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 83148 variables used to determine the values of locale categories.)

- 83149 *LC_ALL* If set to a non-empty string value, override the values of all the other
83150 internationalization variables.
- 83151 *LC_COLLATE*
83152 Determine the locale for the ordering of the output when the *-r* option is used.
- 83153 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
83154 characters (for example, single-byte as opposed to multi-byte characters in
83155 arguments and input files).
- 83156 *LC_MESSAGES*
83157 Determine the locale that should be used to affect the format and contents of
83158 diagnostic messages written to standard error.
- 83159 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 83160 **ASYNCHRONOUS EVENTS**
- 83161 Default.
- 83162 **STDOUT**
- 83163 The flowgraph written to standard output shall be formatted as follows:
- 83164 "%d %s:%s\n", <reference number>, <global>, <definition>
- 83165 Each line of output begins with a reference (that is, line) number, followed by indentation of at
83166 least one column position per level. This is followed by the name of the global, a <colon>, and
83167 its definition. Normally globals are only functions not defined as an external or beginning with
83168 an <underscore>; see the **OPTIONS** section for the *-i* inclusion option. For information extracted
83169 from C-language source, the definition consists of an abstract type declaration (for example, **char**
83170 *) and, delimited by angle brackets, the name of the source file and the line number where the
83171 definition was found. Definitions extracted from object files indicate the filename and location
83172 counter under which the symbol appeared (for example, *text*).
- 83173 Once a definition of a name has been written, subsequent references to that name contain only
83174 the reference number of the line where the definition can be found. For undefined references,
83175 only "<>" shall be written.
- 83176 **STDERR**
- 83177 The standard error shall be used only for diagnostic messages.
- 83178 **OUTPUT FILES**
- 83179 None.
- 83180 **EXTENDED DESCRIPTION**
- 83181 None.
- 83182 **EXIT STATUS**
- 83183 The following exit values shall be returned:
- 83184 0 Successful completion.
- 83185 >0 An error occurred.
- 83186 **CONSEQUENCES OF ERRORS**
- 83187 Default.

83188 APPLICATION USAGE

83189 Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can
83190 confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

83191 EXAMPLES

83192 Given the following in **file.c**:

```
83193     int i;
83194     int f();
83195     int g();
83196     int h();
83197     int
83198     main()
83199     {
83200         f();
83201         g();
83202         f();
83203     }
83204     int
83205     f()
83206     {
83207         i = h();
83208     }
```

83209 The command:

```
83210 cflow -i x file.c
```

83211 produces the output:

```
83212 1 main: int(), <file.c 6>
83213 2   f: int(), <file.c 13>
83214 3     h: <>
83215 4       i: int, <file.c 1>
83216 5     g: <>
```

83217 RATIONALE

83218 None.

83219 FUTURE DIRECTIONS

83220 None.

83221 SEE ALSO

83222 [c99](#), [lex](#), [yacc](#)

83223 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

83224 CHANGE HISTORY

83225 First released in Issue 2.

83226 Issue 6

83227 The normative text is reworded to avoid use of the term “must” for application requirements.

83228 Issue 7

83229 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83230 **NAME**

83231 chgrp — change the file group ownership

83232 **SYNOPSIS**

83233 chgrp [-h] group file...

83234 chgrp -R [-H|-L|-P] group file...

83235 **DESCRIPTION**83236 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID
83237 specified by the *group* operand.83238 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the
83239 directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to
83240 the *chown()* function defined in the System Interfaces volume of POSIX.1-2017, called with the
83241 following arguments:83242 The *file* operand shall be used as the *path* argument.83243 The user ID of the file shall be used as the *owner* argument.83244 The specified group ID shall be used as the *group* argument.83245 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-
83246 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-
83247 group-ID bits of other file types may be cleared.83248 **OPTIONS**83249 The *chgrp* utility shall conform to XBD [Section 12.2](#) (on page 216).

83250 The following options shall be supported by the implementation:

83251 **-h** For each *file* operand that names a file of type symbolic link, *chgrp* shall attempt to
83252 set the group ID of the symbolic link instead of the file referenced by the symbolic
83253 link.83254 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory
83255 is specified on the command line, *chgrp* shall change the group of the directory
83256 referenced by the symbolic link and all files in the file hierarchy below it.83257 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory
83258 is specified on the command line or encountered during the traversal of a file
83259 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic
83260 link and all files in the file hierarchy below it.83261 **-P** If the **-R** option is specified and a symbolic link is specified on the command line
83262 or encountered during the traversal of a file hierarchy, *chgrp* shall change the group
83263 ID of the symbolic link. The *chgrp* utility shall not follow the symbolic link to any
83264 other part of the file hierarchy.83265 **-R** Recursively change file group IDs. For each *file* operand that names a directory,
83266 *chgrp* shall change the group of the directory and all files in the file hierarchy
83267 below it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these
83268 options will be used as the default.83269 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
83270 considered an error. The last option specified shall determine the behavior of the utility.

83271 **OPERANDS**

83272 The following operands shall be supported:

83273 *group* A group name from the group database or a numeric group ID. Either specifies a
 83274 group ID to be given to each file named by one of the *file* operands. If a numeric
 83275 *group* operand exists in the group database as a group name, the group ID number
 83276 associated with that group name is used as the group ID.

83277 *file* A pathname of a file whose group ID is to be modified.

83278 **STDIN**

83279 Not used.

83280 **INPUT FILES**

83281 None.

83282 **ENVIRONMENT VARIABLES**83283 The following environment variables shall affect the execution of *chgrp*:

83284 *LANG* Provide a default value for the internationalization variables that are unset or null.
 83285 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 83286 variables used to determine the values of locale categories.)

83287 *LC_ALL* If set to a non-empty string value, override the values of all the other
 83288 internationalization variables.

83289 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 83290 characters (for example, single-byte as opposed to multi-byte characters in
 83291 arguments).

83292 *LC_MESSAGES*

83293 Determine the locale that should be used to affect the format and contents of
 83294 diagnostic messages written to standard error.

83295 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

83296 **ASYNCHRONOUS EVENTS**

83297 Default.

83298 **STDOUT**

83299 Not used.

83300 **STDERR**

83301 The standard error shall be used only for diagnostic messages.

83302 **OUTPUT FILES**

83303 None.

83304 **EXTENDED DESCRIPTION**

83305 None.

83306 **EXIT STATUS**

83307 The following exit values shall be returned:

83308 0 The utility executed successfully and all requested changes were made.

83309 >0 An error occurred.

83310 **CONSEQUENCES OF ERRORS**

83311 Default.

83312 **APPLICATION USAGE**83313 Only the owner of a file or the user with appropriate privileges may change the owner or group
83314 of a file.83315 Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the
83316 *group* specified is not the effective group ID or one of the supplementary group IDs of the calling
83317 process.83318 **EXAMPLES**

83319 None.

83320 **RATIONALE**83321 The System V and BSD versions use different exit status codes. Some implementations used the
83322 exit status as a count of the number of errors that occurred; this practice is unworkable since it
83323 can overflow the range of valid exit status values. The standard developers chose to mask these
83324 by specifying only 0 and >0 as exit values.83325 The functionality of *chgrp* is described substantially through references to *chown()*. In this way,
83326 there is no duplication of effort required for describing the interactions of permissions, multiple
83327 groups, and so on.83328 **FUTURE DIRECTIONS**

83329 None.

83330 **SEE ALSO**83331 *chmod*, *chown*83332 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)83333 XSH *chown()*83334 **CHANGE HISTORY**

83335 First released in Issue 2.

83336 **Issue 6**83337 New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
83338 options affect the processing of symbolic links.83339 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
83340 section to “Default”.83341 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to
83342 make it clear that **-h** and **-R** are optional.83343 **Issue 7**83344 SD5-XCU-ERN-8 is applied, removing the **-R** from the first line of the SYNOPSIS.

83345 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83346 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0080 [237,341] is applied.

83347 **NAME**

83348 chmod ‡change the file modes

83349 **SYNOPSIS**83350 chmod [-R] *mode file...*83351 **DESCRIPTION**83352 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*
83353 operand in the way specified by the *mode* operand.83354 It is implementation-defined whether and how the *chmod* utility affects any alternate or
83355 additional file access control mechanism (see XBD Section 4.5, on page 108) being used for the
83356 specified file.83357 Only a process whose effective user ID matches the user ID of the file, or a process with
83358 appropriate privileges, shall be permitted to change the file mode bits of a file.83359 Upon successfully changing the file mode bits of a file, the *chmod* utility shall mark for update
83360 the last file status change timestamp of the file.83361 **OPTIONS**83362 The *chmod* utility shall conform to XBD Section 12.2 (on page 216).

83363 The following option shall be supported:

83364 **-R** Recursively change file mode bits. For each *file* operand that names a directory,
83365 *chmod* shall change the file mode bits of the directory and all files in the file
83366 hierarchy below it.83367 **OPERANDS**

83368 The following operands shall be supported:

83369 *mode* Represents the change to be made to the file mode bits of each file named by one of
83370 the *file* operands; see the EXTENDED DESCRIPTION section.83371 *file* A pathname of a file whose file mode bits shall be modified.83372 **STDIN**

83373 Not used.

83374 **INPUT FILES**

83375 None.

83376 **ENVIRONMENT VARIABLES**83377 The following environment variables shall affect the execution of *chmod*:83378 *LANG* Provide a default value for the internationalization variables that are unset or null.
83379 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
83380 variables used to determine the values of locale categories.)83381 *LC_ALL* If set to a non-empty string value, override the values of all the other
83382 internationalization variables.83383 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
83384 characters (for example, single-byte as opposed to multi-byte characters in
83385 arguments).83386 *LC_MESSAGES*83387 Determine the locale that should be used to affect the format and contents of
83388 diagnostic messages written to standard error.

- 83389 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 83390 **ASYNCHRONOUS EVENTS**
- 83391 Default.
- 83392 **STDOUT**
- 83393 Not used.
- 83394 **STDERR**
- 83395 The standard error shall be used only for diagnostic messages.
- 83396 **OUTPUT FILES**
- 83397 None.
- 83398 **EXTENDED DESCRIPTION**
- 83399 The *mode* operand shall be either a *symbolic_mode* expression or a non-negative octal integer. The
- 83400 *symbolic_mode* form is described by the grammar later in this section.
- 83401 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.
- 83402 The operations shall be performed on each *file* in the order in which the **clauses** are specified.
- 83403 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,
- 83404 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.
- 83405 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode
- 83406 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**
- 83407 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.
- 83408 The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a
- 83409 directory or if the current (unmodified) file mode bits have at least one of the execute bits
- 83410 (**S_IXUSR**, **S_IXGRP**, or **S_IXOTH**) set. It shall be ignored if the file is not a directory and none of
- 83411 the execute bits are set in the current file mode bits.
- 83412 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the
- 83413 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,
- 83414 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.
- 83415 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be
- 83416 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation
- 83417 performed, as follows:
- 83418 + If **perm** is not specified, the '+' operation shall not change the file mode bits.
- 83419 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
- 83420 other permissions, except for those with corresponding bits in the file mode creation mask
- 83421 of the invoking process, shall be set.
- 83422 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.
- 83423 - If **perm** is not specified, the '-' operation shall not change the file mode bits.
- 83424 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
- 83425 other permissions, except for those with corresponding bits in the file mode creation mask
- 83426 of the invoking process, shall be cleared.
- 83427 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be
- 83428 cleared.
- 83429 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the
- 83430 file mode bits specified in this volume of POSIX.1-2017.

83431 If **perm** is not specified, the '=' operation shall make no further modifications to the file
83432 mode bits.

83433 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
83434 other permissions, except for those with corresponding bits in the file mode creation mask
83435 of the invoking process, shall be set.

83436 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

83437 When using the symbolic mode form on a regular file, it is implementation-defined whether or
83438 not:

83439 Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all
83440 execute bits are currently clear and none are being set are ignored.

83441 Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-
83442 on-execution bits.

83443 Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all
83444 execute bits are currently clear are ignored. However, if the command *ls -l file* writes an *s*
83445 in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is
83446 set, the commands *chmod u-s file* or *chmod g-s file*, respectively, shall not be ignored.

83447 When using the symbolic mode form on other file types, it is implementation-defined whether
83448 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
83449 honored.

83450 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols
83451 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be
83452 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**
83453 symbol **s**.

83454 XSI The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory, it can
83455 be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who**
83456 symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these
83457 combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other
83458 than directory is unspecified.

83459 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

83460 For each bit set in the octal number, the corresponding file permission bit shown in the following
83461 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in
83462 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-
83463 execution, bits shown in the following table shall be set; if these bits are not set in the octal
83464 number, they are cleared. For other file types, it is implementation-defined whether or not
83465 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
83466 honored.

Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit
4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
1000	S_ISVTX	0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

83467 XSI
83468
83469
83470
83471 When bits are set in the octal number other than those listed in the table above, the behavior is
83472 unspecified.

83473 **Grammar for chmod**

83474 The grammar and lexical conventions in this section describe the syntax for the *symbolic_mode*
 83475 operand. The general conventions for this style of grammar are described in [Section 1.3](#) (on page
 83476 2335). A valid *symbolic_mode* can be represented as the non-terminal symbol *symbolic_mode* in
 83477 the grammar. This formal syntax shall take precedence over the preceding text syntax
 83478 description.

83479 The lexical processing is based entirely on single characters. Implementations need not allow
 83480 <blank> characters within the single argument being processed.

```

83481 %start    symbolic_mode
83482 %%

83483 symbolic_mode    : clause
83484                  | symbolic_mode ',' clause
83485                  ;

83486 clause          : actionlist
83487                  | wholist actionlist
83488                  ;

83489 wholist         : who
83490                  | wholist who
83491                  ;

83492 who             : 'u' | 'g' | 'o' | 'a'
83493                  ;

83494 actionlist      : action
83495                  | actionlist action
83496                  ;

83497 action         : op
83498                  | op permlist
83499                  | op permcopy
83500                  ;

83501 permcopy       : 'u' | 'g' | 'o'
83502                  ;

83503 op             : '+' | '-' | '='
83504                  ;

83505 permlist       : perm
83506                  | perm permlist
83507                  ;

83508 XSI perm       : 'r' | 'w' | 'x' | 'X' | 's' | 't'
83509                  ;

```

83510 **EXIT STATUS**

83511 The following exit values shall be returned:

83512 0 The utility executed successfully and all requested changes were made.

83513 >0 An error occurred.

83514 **CONSEQUENCES OF ERRORS**

83515 Default.

83516 **APPLICATION USAGE**

83517 Some implementations of the *chmod* utility change the mode of a directory before the files in the
 83518 directory when performing a recursive (**-R** option) change; others change the directory mode
 83519 after the files in the directory. If an application tries to remove read or search permission for a
 83520 file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying
 83521 to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users
 83522 should not try to make a hierarchy inaccessible to themselves.

83523 Some implementations of *chmod* never used the *umask* of the process when changing modes;
 83524 systems conformant with this volume of POSIX.1-2017 do so when **who** is not specified. Note
 83525 the difference between:

83526 `chmod a-w file`

83527 which removes all write permissions, and:

83528 `chmod -- -w file`

83529 which removes write permissions that would be allowed if **file** was created with the same
 83530 *umask*.

83531 Conforming applications should never assume that they know how the set-user-ID and set-
 83532 group-ID bits on directories are interpreted.

83533 **EXAMPLES**

83534

Mode	Results
<i>a+=</i>	Equivalent to <i>a+,a=</i> ; clears all file mode bits.
<i>go+-w</i>	Equivalent to <i>go+,go-w</i> ; clears group and other write bits.
<i>g=o-w</i>	Equivalent to <i>g=o,g-w</i> ; sets group bit to match other bits and then clears group write bit.
<i>g-r+w</i>	Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.
<i>uo=g</i>	Sets owner bits to match group bits and sets other bits to match group bits.

83535

83536

83537

83538

83539

83540

83541

83542

83543

83544 **RATIONALE**

83545 The functionality of *chmod* is described substantially through references to concepts defined in
 83546 the System Interfaces volume of POSIX.1-2017. In this way, there is less duplication of effort
 83547 required for describing the interactions of permissions. However, the behavior of this utility is
 83548 not described in terms of the *chmod()* function from the System Interfaces volume of
 83549 POSIX.1-2017 because that specification requires certain side-effects upon alternate file access
 83550 control mechanisms that might not be appropriate, depending on the implementation.

83551 Implementations that support mandatory file and record locking as specified by the 1984
 83552 /usr/group standard historically used the combination of set-group-ID bit set and group
 83553 execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the
 83554 symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory
 83555 locking mode is not changed without explicit indication that that was what the user intended.
 83556 Therefore, the details on how the implementation treats these conditions must be defined in the
 83557 documentation. This volume of POSIX.1-2017 does not require mandatory locking (nor does the
 83558 System Interfaces volume of POSIX.1-2017), but does allow it as an extension. However, this
 83559 volume of POSIX.1-2017 does require that the *ls* and *chmod* utilities work consistently in this

83560 area. If `ls -l file` indicates that the set-group-ID bit is set, `chmod g-s file` must clear it (assuming
83561 appropriate privileges exist to change modes).

83562 The System V and BSD versions use different exit status codes. Some implementations used the
83563 exit status as a count of the number of errors that occurred; this practice is unworkable since it
83564 can overflow the range of valid exit status values. This problem is avoided here by specifying
83565 only 0 and >0 as exit values.

83566 The System Interfaces volume of POSIX.1-2017 indicates that implementation-defined
83567 restrictions may cause the S_ISUID and S_ISGID bits to be ignored. This volume of POSIX.1-2017
83568 allows the `chmod` utility to choose to modify these bits before calling `chmod()` (or some function
83569 providing equivalent capabilities) for non-regular files. Among other things, this allows
83570 implementations that use the set-user-ID and set-group-ID bits on directories to enable extended
83571 features to handle these extensions in an intelligent manner.

83572 The **X perm** symbol was adopted from BSD-based systems because it provides commonly
83573 desired functionality when doing recursive (`-R` option) modifications. Similar functionality is
83574 not provided by the `find` utility. Historical BSD versions of `chmod`, however, only supported **X**
83575 with `op+`; it has been extended in this volume of POSIX.1-2017 because it is also useful with `op=`.
83576 (It has also been added for `op-` even though it duplicates **x**, in this case, because it is intuitive
83577 and easier to explain.)

83578 The grammar was extended with the `permcopy` non-terminal to allow historical-practice forms of
83579 symbolic modes like `o=u -g` (that is, set the “other” permissions to the permissions of “owner”
83580 minus the permissions of “group”).

83581 FUTURE DIRECTIONS

83582 None.

83583 SEE ALSO

83584 [*ls*](#), [*umask*](#)

83585 XBD [Section 4.5](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

83586 XSH `chmod()`

83587 CHANGE HISTORY

83588 First released in Issue 2.

83589 Issue 6

83590 The following new requirements on POSIX implementations derive from alignment with the
83591 Single UNIX Specification:

83592 Octal modes have been kept and made mandatory despite being marked obsolescent in the
83593 ISO POSIX-2: 1993 standard.

83594 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
83595 section to “Default.”.

83596 The Open Group Base Resolution bwg2001-010 is applied, adding the description of the
83597 S_ISVTX bit and the **t perm** symbol as part of the XSI option.

83598 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded
83599 text in the EXTENDED DESCRIPTION from:

83600 “The **perm** symbol **t** shall specify the S_ISVTX bit and shall apply to directories only. The
83601 effect when using it with any other file type is unspecified. It can be used with the **who**
83602 symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**
83603 or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.”

83604 to:
83605 `The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory,
83606 it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to
83607 specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning
83608 of these combinations is unspecified. The effect when using the **perm** symbol **t** with any
83609 file type other than directory is unspecified.”

83610 This change is to permit historical behavior.

83611 **Issue 7**

83612 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83613 Austin Group Interpretation 1003.1-2001 #130 is applied, adding text to the DESCRIPTION
83614 about about marking for update the last file status change timestamp of the file.

83615 **NAME**

83616 chown ‡change the file ownership

83617 **SYNOPSIS**

83618 chown [-h] owner[:group] file...

83619 chown -R [-H|-L|-P] owner[:group] file...

83620 **DESCRIPTION**83621 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID
83622 specified by the *owner* operand.83623 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the
83624 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to
83625 the *chown()* function defined in the System Interfaces volume of POSIX.1-2017, called with the
83626 following arguments:

- 83627 1. The *file* operand shall be used as the *path* argument.
- 83628 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*
83629 argument.
- 83630 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used
83631 as the *group* argument; otherwise, the group ownership shall not be changed.

83632 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-group-
83633 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-
83634 group-ID bits of other file types may be cleared.83635 **OPTIONS**83636 The *chown* utility shall conform to XBD [Section 12.2](#) (on page 216).

83637 The following options shall be supported by the implementation:

- 83638 **-h** For each file operand that names a file of type symbolic link, *chown* shall attempt to
83639 set the user ID of the symbolic link. If a group ID was specified, for each file
83640 operand that names a file of type symbolic link, *chown* shall attempt to set the
83641 group ID of the symbolic link.
- 83642 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory
83643 is specified on the command line, *chown* shall change the user ID (and group ID, if
83644 specified) of the directory referenced by the symbolic link and all files in the file
83645 hierarchy below it.
- 83646 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory
83647 is specified on the command line or encountered during the traversal of a file
83648 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the
83649 directory referenced by the symbolic link and all files in the file hierarchy below it.
- 83650 **-P** If the **-R** option is specified and a symbolic link is specified on the command line
83651 or encountered during the traversal of a file hierarchy, *chown* shall change the
83652 owner ID (and group ID, if specified) of the symbolic link. The *chown* utility shall
83653 not follow the symbolic link to any other part of the file hierarchy.
- 83654 **-R** Recursively change file user and group IDs. For each *file* operand that names a
83655 directory, *chown* shall change the user ID (and group ID, if specified) of the
83656 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is
83657 specified, it is unspecified which of these options will be used as the default.

83658 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be

83659 considered an error. The last option specified shall determine the behavior of the utility.

83660 OPERANDS

83661 The following operands shall be supported:

83662 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The *owner* portion of this
 83663 operand shall be a user name from the user database or a numeric user ID. Either
 83664 specifies a user ID which shall be given to each file named by one of the *file*
 83665 operands. If a numeric *owner* operand exists in the user database as a user name,
 83666 the user ID number associated with that user name shall be used as the user ID.
 83667 Similarly, if the *group* portion of this operand is present, it shall be a group name
 83668 from the group database or a numeric group ID. Either specifies a group ID which
 83669 shall be given to each file. If a numeric group operand exists in the group database
 83670 as a group name, the group ID number associated with that group name shall be
 83671 used as the group ID.

83672 *file* A pathname of a file whose user ID is to be modified.

83673 STDIN

83674 Not used.

83675 INPUT FILES

83676 None.

83677 ENVIRONMENT VARIABLES

83678 The following environment variables shall affect the execution of *chown*:

83679 *LANG* Provide a default value for the internationalization variables that are unset or null.
 83680 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 83681 variables used to determine the values of locale categories.)

83682 *LC_ALL* If set to a non-empty string value, override the values of all the other
 83683 internationalization variables.

83684 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 83685 characters (for example, single-byte as opposed to multi-byte characters in
 83686 arguments).

83687 *LC_MESSAGES*

83688 Determine the locale that should be used to affect the format and contents of
 83689 diagnostic messages written to standard error.

83690 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

83691 ASYNCHRONOUS EVENTS

83692 Default.

83693 STDOUT

83694 Not used.

83695 STDERR

83696 The standard error shall be used only for diagnostic messages.

83697 OUTPUT FILES

83698 None.

83699 **EXTENDED DESCRIPTION**

83700 None.

83701 **EXIT STATUS**

83702 The following exit values shall be returned:

83703 0 The utility executed successfully and all requested changes were made.

83704 >0 An error occurred.

83705 **CONSEQUENCES OF ERRORS**

83706 Default.

83707 **APPLICATION USAGE**83708 Only the owner of a file or the user with appropriate privileges may change the owner or group
83709 of a file.83710 Some implementations restrict the use of *chown* to a user with appropriate privileges.83711 **EXAMPLES**

83712 None.

83713 **RATIONALE**83714 The System V and BSD versions use different exit status codes. Some implementations used the
83715 exit status as a count of the number of errors that occurred; this practice is unworkable since it
83716 can overflow the range of valid exit status values. These are masked by specifying only 0 and >0
83717 as exit values.83718 The functionality of *chown* is described substantially through references to functions in the
83719 System Interfaces volume of POSIX.1-2017. In this way, there is no duplication of effort required
83720 for describing the interactions of permissions, multiple groups, and so on.83721 The 4.3 BSD method of specifying both owner and group was included in this volume of
83722 POSIX.1-2017 because:83723 There are cases where the desired end condition could not be achieved using the *chgrp* and
83724 *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the
83725 desired group and the desired owner is not a member of the current group, the *chown()*
83726 function could fail unless both owner and group are changed at the same time.)83727 Even if they could be changed independently, in cases where both are being changed, there
83728 is a 100% performance penalty caused by being forced to invoke both utilities.83729 The BSD syntax *user[.group]* was changed to *user[:group]* in this volume of POSIX.1-2017 because
83730 the <period> is a valid character in login names (as specified by the Base Definitions volume of
83731 POSIX.1-2017, login names consist of characters in the portable filename character set). The
83732 <colon> character was chosen as the replacement for the <period> character because it would
83733 never be allowed as a character in a user name or group name on historical implementations.83734 The **-R** option is considered by some observers as an undesirable departure from the historical
83735 UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there
83736 seemed to be no good reason to require other tools to have to duplicate that functionality.
83737 However, the **-R** option was deemed an important user convenience, is far more efficient than
83738 forking a separate process for each element of the directory hierarchy, and is in widespread
83739 historical use.

83740 **FUTURE DIRECTIONS**

83741 None.

83742 **SEE ALSO**83743 *chgrp, chmod*83744 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)83745 XSH *chown()*83746 **CHANGE HISTORY**

83747 First released in Issue 2.

83748 **Issue 6**83749 New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
83750 options affect the processing of symbolic links.

83751 The normative text is reworded to avoid use of the term “must” for application requirements.

83752 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
83753 section to “Default.”.83754 The “otherwise, ...” text in item 3. of the DESCRIPTION is changed to “otherwise, the group
83755 ownership shall not be changed”.83756 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to
83757 make it clear that **-h** and **-R** are optional.83758 **Issue 7**83759 SD5-XCU-ERN-9 is applied, removing the **-R** from the first line of the SYNOPSIS.

83760 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83761 The description of the **-h** and **-P** options is revised.

83762 **NAME**

83763 cksum ‡write file checksums and sizes

83764 **SYNOPSIS**83765 cksum [*file*...]83766 **DESCRIPTION**

83767 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)
 83768 for each input file, and also write to standard output the number of octets in each file. The CRC
 83769 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996
 83770 standard (Ethernet).

83771 The encoding for the CRC checksum is defined by the generating polynomial:

$$83772 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

83773 Mathematically, the CRC value corresponding to a given file shall be defined by the following
 83774 procedure:

- 83775 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial
 83776 *M(x)* of degree *n*−1. These *n* bits are the bits from the file, with the most significant bit
 83777 being the most significant bit of the first octet of the file and the last bit being the least
 83778 significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral
 83779 number of octets, followed by one or more octets representing the length of the file as a
 83780 binary value, least significant octet first. The smallest number of octets capable of
 83781 representing this integer shall be used.
- 83782 2. *M(x)* is multiplied by x^{32} (that is, shifted left 32 bits) and divided by *G(x)* using mod 2
 83783 division, producing a remainder *R(x)* of degree ≤ 31.
- 83784 3. The coefficients of *R(x)* are considered to be a 32-bit sequence.
- 83785 4. The bit sequence is complemented and the result is the CRC.

83786 **OPTIONS**

83787 None.

83788 **OPERANDS**

83789 The following operand shall be supported:

83790 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard
 83791 input shall be used.

83792 **STDIN**

83793 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
 83794 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
 83795 the standard input shall not be used. See the INPUT FILES section.

83796 **INPUT FILES**

83797 The input files can be any file type.

83798 **ENVIRONMENT VARIABLES**83799 The following environment variables shall affect the execution of *cksum*:

83800 *LANG* Provide a default value for the internationalization variables that are unset or null.
 83801 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 83802 variables used to determine the values of locale categories.)

83803 *LC_ALL* If set to a non-empty string value, override the values of all the other
 83804 internationalization variables.

83805 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 83806 characters (for example, single-byte as opposed to multi-byte characters in
 83807 arguments).

83808 *LC_MESSAGES*
 83809 Determine the locale that should be used to affect the format and contents of
 83810 diagnostic messages written to standard error.

83811 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

83812 ASYNCHRONOUS EVENTS

83813 Default.

83814 STDOUT

83815 For each file processed successfully, the *cksum* utility shall write in the following format:

83816 "%u %d %s\n", <checksum>, <# of octets>, <pathname>

83817 If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

83818 STDERR

83819 The standard error shall be used only for diagnostic messages.

83820 OUTPUT FILES

83821 None.

83822 EXTENDED DESCRIPTION

83823 None.

83824 EXIT STATUS

83825 The following exit values shall be returned:

83826 0 All files were processed successfully.

83827 >0 An error occurred.

83828 CONSEQUENCES OF ERRORS

83829 Default.

83830 APPLICATION USAGE

83831 The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of
 83832 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this
 83833 comparison cannot be considered cryptographically secure. The chances of a damaged file
 83834 producing the same CRC as the original are small; deliberate deception is difficult, but probably
 83835 not impossible.

83836 Although input files to *cksum* can be any type, the results need not be what would be expected
 83837 on character special device files or on file types not described by the System Interfaces volume of
 83838 POSIX.1-2017. Since this volume of POSIX.1-2017 does not specify the block size used when
 83839 doing input, checksums of character special files need not process all of the data in those files.

83840 The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted
 83841 between two systems and undergoes any data transformation (such as changing little-endian
 83842 byte ordering to big-endian), identical CRC values cannot be expected. Implementations
 83843 performing such transformations may extend *cksum* to handle such situations.

83844 EXAMPLES

83845 None.

83846 **RATIONALE**

83847 The following C-language program can be used as a model to describe the algorithm. It assumes
 83848 that a **char** is one octet. It also assumes that the entire file is available for one pass through the
 83849 function. This was done for simplicity in demonstrating the algorithm, rather than as an
 83850 implementation model.

```

83851 static unsigned long crctab[] = {
83852     0x00000000,
83853     0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
83854     0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
83855     0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbdbd,
83856     0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
83857     0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
83858     0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
83859     0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
83860     0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
83861     0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
83862     0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
83863     0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
83864     0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
83865     0xe5ffe43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
83866     0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
83867     0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcdbb16,
83868     0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
83869     0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93dadb, 0x6f52c06c,
83870     0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
83871     0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
83872     0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b,
83873     0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
83874     0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
83875     0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
83876     0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
83877     0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
83878     0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcdfd59, 0x608edb80,
83879     0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
83880     0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
83881     0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
83882     0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
83883     0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
83884     0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
83885     0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
83886     0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
83887     0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
83888     0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
83889     0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
83890     0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
83891     0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
83892     0x4e8ee645, 0x4a4ffb2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
83893     0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
83894     0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
83895     0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
83896     0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
  
```



```

83897     0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
83898     0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
83899     0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xee2ed18,
83900     0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
83901     0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
83902     0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
83903     0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
83904     };

83905     unsigned long memcrc(const unsigned char *b, size_t n)
83906     {
83907     /*   Input arguments:
83908        *   const unsigned char*   b == byte sequence to checksum
83909        *   size_t                   n == length of sequence
83910        */

83911         register size_t i;
83912         register unsigned c, s = 0;

83913         for (i = n; i > 0; --i) {
83914             c = *b++;
83915             s = (s << 8) ^ crctab[(s >> 24) ^ c];
83916         }

83917         /* Extend with the length of the string. */
83918         while (n != 0) {
83919             c = n & 0377;
83920             n >>= 8;
83921             s = (s << 8) ^ crctab[(s >> 24) ^ c];
83922         }

83923         return ~s;
83924     }

```

83925 The historical practice of writing the number of “blocks” has been changed to writing the
83926 number of octets, since the latter is not only more useful, but also since historical
83927 implementations have not been consistent in defining what a “block” meant.

83928 The algorithm used was selected to increase the operational robustness of *cksum*. Neither the
83929 System V nor BSD *sum* algorithm was selected. Since each of these was different and each was
83930 the default behavior on those systems, no realistic compromise was available if either were
83931 selected—some set of historical applications would break. Therefore, the name was changed to
83932 *cksum*. Although the historical *sum* commands will probably continue to be provided for many
83933 years, programs designed for portability across systems should use the new name.

83934 The algorithm selected is based on that used by the ISO/IEC 8802-3:1996 standard (Ethernet) for
83935 the frame check sequence field. The algorithm used does not match the technical definition of a
83936 *checksum*; the term is used for historical reasons. The length of the file is included in the CRC
83937 calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also
83938 because it guards against inadvertent collisions between files that begin with different series of
83939 zero octets. The chance that two different files produce identical CRCs is much greater when
83940 their lengths are not considered. Keeping the length and the checksum of the file itself separate
83941 would yield a slightly more robust algorithm, but historical usage has always been that a single
83942 number (the checksum as printed) represents the signature of the file. It was decided that
83943 historical usage was the more important consideration.

83944 Early proposals contained modifications to the Ethernet algorithm that involved extracting table
 83945 values whenever an intermediate result became zero. This was demonstrated to be less robust
 83946 than the current method and mathematically difficult to describe or justify.

83947 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.
 83948 The pseudo-code rendition is:

```
83949 X <- 0; Y <- 0;
83950 for i <- m -1 step -1 until 0 do
83951   begin
83952     T <- X(1) ^ A[i];
83953     X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
83954     comment: f[T] and f'[T] denote the T-th words in the
83955             table f and f' ;
83956     X <- X ^ f[T]; Y <- Y ^ f'[T];
83957   end
```

83958 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**
 83959 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables
 83960 **f** and **f'** are a single table containing 32-bit values.

83961 The referenced Sarwate article also discusses generating the table.

83962 **FUTURE DIRECTIONS**

83963 None.

83964 **SEE ALSO**

83965 XBD [Chapter 8](#) (on page 173)

83966 **CHANGE HISTORY**

83967 First released in Issue 4.

83968 **Issue 7**

83969 Austin Group Interpretation 1003.1-2001 #092 is applied.

83970 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

83971 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0081 [446] is applied.

83972 **NAME**

83973 cmp ‡'compare two files

83974 **SYNOPSIS**83975 cmp [-l|-s] *file1 file2*83976 **DESCRIPTION**

83977 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the
 83978 same. Under default options, if they differ, it shall write to standard output the byte and line
 83979 number at which the first difference occurred. Bytes and lines shall be numbered beginning with
 83980 1.

83981 **OPTIONS**83982 The *cmp* utility shall conform to XBD [Section 12.2](#) (on page 216).

83983 The following options shall be supported:

83984 **-l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for
 83985 each difference.

83986 **-s** Write nothing to standard output or standard error when files differ; indicate
 83987 differing files through exit status only. It is unspecified whether a diagnostic
 83988 message is written to standard error when an error is encountered; if a message is
 83989 not written, the error is indicated through exit status only.

83990 **OPERANDS**

83991 The following operands shall be supported:

83992 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall
 83993 be used.

83994 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input
 83995 shall be used.

83996 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or
 83997 character special file, the results are undefined.

83998 **STDIN**

83999 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the
 84000 INPUT FILES section.

84001 **INPUT FILES**

84002 The input files can be any file type.

84003 **ENVIRONMENT VARIABLES**84004 The following environment variables shall affect the execution of *cmp*:

84005 **LANG** Provide a default value for the internationalization variables that are unset or null.
 84006 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 84007 variables used to determine the values of locale categories.)

84008 **LC_ALL** If set to a non-empty string value, override the values of all the other
 84009 internationalization variables.

84010 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 84011 characters (for example, single-byte as opposed to multi-byte characters in
 84012 arguments).

84013 **LC_MESSAGES**

84014 Determine the locale that should be used to affect the format and contents of
 84015 diagnostic messages written to standard error and informative messages written to

84016 standard output.

84017 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

84018 **ASYNCHRONOUS EVENTS**

84019 Default.

84020 **STDOUT**

84021 In the POSIX locale, results of the comparison shall be written to standard output. When no
84022 options are used, the format shall be:

84023 "%s %s differ: char %d, line %d\n", *file1*, *file2*,
84024 <*byte number*>, <*line number*>

84025 When the **-l** option is used, the format shall be:

84026 "%d %o %o\n", <*byte number*>, <*differing byte*>,
84027 <*differing byte*>

84028 for each byte that differs. The first <*differing byte*> number is from *file1* while the second is from
84029 *file2*. In both cases, <*byte number*> shall be relative to the beginning of the file, beginning with 1.

84030 No output shall be written to standard output when the **-s** option is used.

84031 **STDERR**

84032 The standard error shall be used only for diagnostic messages. If the **-l** option is used and *file1*
84033 and *file2* differ in length, or if the **-s** option is not used and *file1* and *file2* are identical for the
84034 entire length of the shorter file, in the POSIX locale the following diagnostic message shall be
84035 written:

84036 "cmp: EOF on %s%s\n", <*name of shorter file*>, <*additional info*>

84037 The <*additional info*> field shall either be null or a string that starts with a <blank> and contains
84038 no <newline> characters. Some implementations report on the number of lines in this case.

84039 If the **-s** option is used and an error occurs, it is unspecified whether a diagnostic message is
84040 written to standard error.

84041 **OUTPUT FILES**

84042 None.

84043 **EXTENDED DESCRIPTION**

84044 None.

84045 **EXIT STATUS**

84046 The following exit values shall be returned:

84047 0 The files are identical.

84048 1 The files are different; this includes the case where one file is identical to the first part of the
84049 other.

84050 >1 An error occurred.

84051 **CONSEQUENCES OF ERRORS**

84052 Default.

84053 **APPLICATION USAGE**

84054 Although input files to *cmp* can be any type, the results might not be what would be expected on
 84055 character special device files or on file types not described by the System Interfaces volume of
 84056 POSIX.1-2017. Since this volume of POSIX.1-2017 does not specify the block size used when
 84057 doing input, comparisons of character special files need not compare all of the data in those files.

84058 For files which are not text files, line numbers simply reflect the presence of a <newline>,
 84059 without any implication that the file is organized into lines.

84060 Since the behavior of `-s` differs between implementations as to whether error messages are
 84061 written, the only way to ensure consistent behavior of *cmp* when `-s` is used is to redirect
 84062 standard error to `/dev/null`.

84063 If error messages are wanted, instead of using `-s` standard output should be redirected to
 84064 `/dev/null`, and anything written to standard error should be discarded if the exit status is 1. For
 84065 example:

```
84066 silent_cmp() {
84067     # compare files with no output except error messages
84068     message=$(cmp "$@" 2>&1 >/dev/null)
84069     status=$?
84070     case $status in
84071         (0|1) ;;
84072         (*) printf '%s\n' "$message" ;;
84073     esac
84074     return $status
84075 }
```

84076 **EXAMPLES**

84077 None.

84078 **RATIONALE**

84079 The global language in [Section 1.4](#) (on page 2336) indicates that using two mutually-exclusive
 84080 options together produces unspecified results. Some System V implementations consider the
 84081 option usage:

```
84082 cmp -l -s ...
```

84083 to be an error. They also treat:

```
84084 cmp -s -l ...
```

84085 as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

84086 The word **char** in the standard output format comes from historical usage, even though it is
 84087 actually a byte number. When *cmp* is supported in other locales, implementations are
 84088 encouraged to use the word *byte* or its equivalent in another language. Users should not
 84089 interpret this difference to indicate that the functionality of the utility changed between locales.

84090 Some implementations report on the number of lines in the identical-but-shorter file case. This is
 84091 allowed by the inclusion of the <additional info> fields in the output format. The restriction on
 84092 having a leading <blank> and no <newline> characters is to make parsing for the filename
 84093 easier. It is recognized that some filenames containing white-space characters make parsing
 84094 difficult anyway, but the restriction does aid programs used on systems where the names are
 84095 predominantly well behaved.

84096 **FUTURE DIRECTIONS**

84097 Future versions of this standard may require that diagnostic messages are written to standard
84098 error when the `-s` option is specified.

84099 **SEE ALSO**

84100 *comm*, *diff*

84101 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

84102 **CHANGE HISTORY**

84103 First released in Issue 2.

84104 **Issue 7**

84105 SD5-XCU-ERN-96 is applied, updating the STDERR section.

84106 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84107 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0075 [478] is applied.

84108 **NAME**

84109 comm — select or reject lines common to two files

84110 **SYNOPSIS**84111 comm [-123] *file1 file2*84112 **DESCRIPTION**84113 The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating
84114 sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and
84115 lines in both files.84116 If the lines in both files are not ordered according to the collating sequence of the current locale,
84117 the results are unspecified.84118 If the collating sequence of the current locale does not have a total ordering of all characters (see
84119 XBD [Section 7.3.2](#), on page 147) and any lines from the input files collate equally but are not
84120 identical, *comm* should treat them as different lines but may treat them as being the same. If it
84121 treats them as different, *comm* should expect them to be ordered according to a further byte-by-
84122 byte comparison using the collating sequence for the POSIX locale and if they are not ordered in
84123 this way, the output of *comm* can identify such lines as being both unique to *file1* and unique to
84124 *file2* instead of being in both files.84125 **OPTIONS**84126 The *comm* utility shall conform to XBD [Section 12.2](#) (on page 216).

84127 The following options shall be supported:

- 84128 -1 Suppress the output column of lines unique to *file1*.
- 84129 -2 Suppress the output column of lines unique to *file2*.
- 84130 -3 Suppress the output column of lines duplicated in *file1* and *file2*.

84131 **OPERANDS**

84132 The following operands shall be supported:

- 84133 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall
84134 be used.
- 84135 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input
84136 shall be used.

84137 If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or
84138 character special file, the results are undefined.84139 **STDIN**84140 The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.
84141 See the INPUT FILES section.84142 **INPUT FILES**

84143 The input files shall be text files.

84144 **ENVIRONMENT VARIABLES**84145 The following environment variables shall affect the execution of *comm*:

- 84146 *LANG* Provide a default value for the internationalization variables that are unset or null.
84147 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
84148 variables used to determine the values of locale categories.)

84149 *LC_ALL* If set to a non-empty string value, override the values of all the other
84150 internationalization variables.

84151 *LC_COLLATE*
84152 Determine the locale for the collating sequence *comm* expects to have been used
84153 when the input files were sorted.

84154 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84155 characters (for example, single-byte as opposed to multi-byte characters in
84156 arguments and input files).

84157 *LC_MESSAGES*
84158 Determine the locale that should be used to affect the format and contents of
84159 diagnostic messages written to standard error.

84160 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

84161 ASYNCHRONOUS EVENTS

84162 Default.

84163 STDOUT

84164 The *comm* utility shall produce output depending on the options selected. If the *-1*, *-2*, and *-3*
84165 options are all selected, *comm* shall write nothing to standard output.

84166 If the *-1* option is not selected, lines contained only in *file1* shall be written using the format:

84167 "%s\n", <line in file1>

84168 If the *-2* option is not selected, lines contained only in *file2* are written using the format:

84169 "%s%s\n", <lead>, <line in file2>

84170 where the string <lead> is as follows:

84171 <tab> The *-1* option is not selected.

84172 null string The *-1* option is selected.

84173 If the *-3* option is not selected, lines contained in both files shall be written using the format:

84174 "%s%s\n", <lead>, <line in both>

84175 where the string <lead> is as follows:

84176 <tab><tab> Neither the *-1* nor the *-2* option is selected.

84177 <tab> Exactly one of the *-1* and *-2* options is selected.

84178 null string Both the *-1* and *-2* options are selected.

84179 If the input files were ordered according to the collating sequence of the current locale, the lines
84180 written shall be in the collating sequence of the current locale. If the input files contained any
84181 lines that collated equally but were not identical and within each file those lines were ordered
84182 according to a further byte-by-byte comparison using the collating sequence for the POSIX
84183 locale, and *comm* treated them as different lines, then lines written that collate equally but are not
84184 identical should be ordered according to a further byte-by-byte comparison using the collating
84185 sequence for the POSIX locale.

84186 STDERR

84187 The standard error shall be used only for diagnostic messages.

84188 **OUTPUT FILES**

84189 None.

84190 **EXTENDED DESCRIPTION**

84191 None.

84192 **EXIT STATUS**

84193 The following exit values shall be returned:

84194 0 All input files were successfully output as specified.

84195 >0 An error occurred.

84196 **CONSEQUENCES OF ERRORS**

84197 Default.

84198 **APPLICATION USAGE**84199 If the input files are not properly presorted, the output of *comm* might not be useful.

84200 When using *comm* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE
 84201 and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte
 84202 sequences that do not form valid characters in some locales, in which case the utility's behavior
 84203 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore
 84204 this problem is avoided.

84205 If the collating sequence of the current locale does not have a total ordering of all characters, this
 84206 can affect the behavior of *comm* in the following ways:

84207 If *comm* treats lines as being the same only if they are identical, some lines can be
 84208 misleadingly identified as being both unique to *file1* and unique to *file2*.

84209 If *comm* treats lines as being the same if they collate equally and a line from *file1* collates
 84210 equally with a line from *file2* but is not identical to it, one of the lines is misleadingly
 84211 identified as being in both files and the other is not written to the output at all.

84212 Such problems can be avoided by forcing the use of the POSIX locale; for example, the following
 84213 identifies lines in both *file1* and *file2*:

```
84214 LC_ALL=POSIX sort file1 > file1.posix
84215 LC_ALL=POSIX sort file2 > file2.posix
84216 LC_ALL=POSIX comm -12 file1.posix file2.posix | sort
```

84217 The final *sort* re-sorts the output of *comm* according to the collating sequence of the original
 84218 locale. Doing this might be difficult if more than one column is output and leading <blank>s
 84219 cannot be ignored.

84220 **EXAMPLES**

84221 If a file named **xcu** contains a sorted list of the utilities in this volume of POSIX.1-2017, a file
 84222 named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide, Issue
 84223 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface
 84224 Definition Third Edition:

```
84225 comm -23 xcu xpg3 | comm -23 - svid89
```

84226 would print a list of utilities in this volume of POSIX.1-2017 not specified by either of the other
 84227 documents:

```
84228 comm -12 xcu xpg3 | comm -12 - svid89
```

84229 would print a list of utilities specified by all three documents, and:

- 84230 `comm -12 xpg3 svid89 | comm -23 - xcu`
- 84231 would print a list of utilities specified by both XPG3 and the SVID, but not specified in this
84232 volume of POSIX.1-2017.
- 84233 **RATIONALE**
- 84234 None.
- 84235 **FUTURE DIRECTIONS**
- 84236 A future version of this standard may require that if any lines from the input files collate equally
84237 but are not identical, then *comm* treats them as different lines and expects them to be ordered
84238 according to a further byte-by-byte comparison using the collating sequence for the POSIX
84239 locale.
- 84240 A future version of this standard may require that if the input files contained any lines that
84241 collated equally but were not identical and within each file those lines were ordered according to
84242 a further byte-by-byte comparison using the collating sequence for the POSIX locale, then lines
84243 written that collate equally but are not identical are ordered according to a further byte-by-byte
84244 comparison using the collating sequence for the POSIX locale.
- 84245 **SEE ALSO**
- 84246 *cmp, diff, sort, uniq*
- 84247 XBD [Section 7.3.2](#) (on page 147), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 84248 **CHANGE HISTORY**
- 84249 First released in Issue 2.
- 84250 **Issue 6**
- 84251 The normative text is reworded to avoid use of the term “must” for application requirements.
- 84252 **Issue 7**
- 84253 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0076 [963], XCU/TC2-2008/0077
84254 [663], and XCU/TC2-2008/0078 [963] are applied.

84255 **NAME**84256 command \ddagger execute a simple command84257 **SYNOPSIS**84258 command [-p] *command_name* [*argument...*]84259 command [-p] [-v|-V] *command_name*84260 **DESCRIPTION**84261 The *command* utility shall cause the shell to treat the arguments as a simple command,
84262 suppressing the shell function lookup that is described in [Section 2.9.1.1](#) (on page 2367), item 1b.84263 If the *command_name* is the same as the name of one of the special built-in utilities, the special
84264 properties in the enumerated list at the beginning of [Section 2.14](#) (on page 2384) shall not occur.
84265 In every other respect, if *command_name* is not the name of a function, the effect of *command*
84266 (with no options) shall be the same as omitting *command*.84267 When the *-v* or *-V* option is used, the *command* utility shall provide information concerning
84268 how a command name is interpreted by the shell.84269 **OPTIONS**84270 The *command* utility shall conform to XBD [Section 12.2](#) (on page 216).

84271 The following options shall be supported:

84272 **-p** Perform the command search using a default value for *PATH* that is guaranteed to
84273 find all of the standard utilities.84274 **-v** Write a string to standard output that indicates the pathname or command that
84275 will be used by the shell, in the current shell execution environment (see [Section](#)
84276 [2.12](#), on page 2381), to invoke *command_name*, but do not invoke *command_name*.84277 Utilities, regular built-in utilities, *command_names* including a <slash>
84278 character, and any implementation-defined functions that are found using
84279 the *PATH* variable (as described in [Section 2.9.1.1](#), on page 2367), shall be
84280 written as absolute pathnames.84281 Shell functions, special built-in utilities, regular built-in utilities not
84282 associated with a *PATH* search, and shell reserved words shall be written as
84283 just their names.84284 An alias shall be written as a command line that represents its alias
84285 definition.84286 Otherwise, no output shall be written and the exit status shall reflect that the
84287 name was not found.84288 **-V** Write a string to standard output that indicates how the name given in the
84289 *command_name* operand will be interpreted by the shell, in the current shell
84290 execution environment (see [Section 2.12](#), on page 2381), but do not invoke
84291 *command_name*. Although the format of this string is unspecified, it shall indicate
84292 in which of the following categories *command_name* falls and shall include the
84293 information stated:84294 Utilities, regular built-in utilities, and any implementation-defined functions
84295 that are found using the *PATH* variable (as described in [Section 2.9.1.1](#), on
84296 page 2367), shall be identified as such and include the absolute pathname in
84297 the string.

- 84298 Other shell functions shall be identified as functions.
- 84299 Aliases shall be identified as aliases and their definitions included in the
84300 string.
- 84301 Special built-in utilities shall be identified as special built-in utilities.
- 84302 Regular built-in utilities not associated with a *PATH* search shall be identified
84303 as regular built-in utilities. (The term “regular” need not be used.)
- 84304 Shell reserved words shall be identified as reserved words.

84305 OPERANDS

84306 The following operands shall be supported:

- 84307 *argument* One of the strings treated as an argument to *command_name*.
- 84308 *command_name*
84309 The name of a utility or a special built-in utility.

84310 STDIN

84311 Not used.

84312 INPUT FILES

84313 None.

84314 ENVIRONMENT VARIABLES

84315 The following environment variables shall affect the execution of *command*:

- 84316 *LANG* Provide a default value for the internationalization variables that are unset or null.
84317 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
84318 variables used to determine the values of locale categories.)
- 84319 *LC_ALL* If set to a non-empty string value, override the values of all the other
84320 internationalization variables.
- 84321 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84322 characters (for example, single-byte as opposed to multi-byte characters in
84323 arguments).
- 84324 *LC_MESSAGES*
84325 Determine the locale that should be used to affect the format and contents of
84326 diagnostic messages written to standard error and informative messages written to
84327 standard output.
- 84328 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 84329 *PATH* Determine the search path used during the command search described in [Section](#)
84330 [2.9.1.1](#) (on page 2367), except as described under the *-p* option.

84331 ASYNCHRONOUS EVENTS

84332 Default.

84333 STDOUT

84334 When the *-v* option is specified, standard output shall be formatted as:

84335 "%s\n", <pathname or command>

84336 When the *-V* option is specified, standard output shall be formatted as:

84337 "%s\n", <unspecified>

84338 **STDERR**

84339 The standard error shall be used only for diagnostic messages.

84340 **OUTPUT FILES**

84341 None.

84342 **EXTENDED DESCRIPTION**

84343 None.

84344 **EXIT STATUS**

84345 When the `-v` or `-V` options are specified, the following exit values shall be returned:

84346 0 Successful completion.

84347 >0 The *command_name* could not be found or an error occurred.

84348 Otherwise, the following exit values shall be returned:

84349 126 The utility specified by *command_name* was found but could not be invoked.

84350 127 An error occurred in the *command* utility or the utility specified by *command_name* could not
84351 be found.

84352 Otherwise, the exit status of *command* shall be that of the simple command specified by the
84353 arguments to *command*.

84354 **CONSEQUENCES OF ERRORS**

84355 Default.

84356 **APPLICATION USAGE**

84357 The order for command search allows functions to override regular built-ins and path searches.
84358 This utility is necessary to allow functions that have the same name as a utility to call the utility
84359 (instead of a recursive call to the function).

84360 The system default path is available using *getconf*; however, since *getconf* may need to have the
84361 *PATH* set up before it can be called itself, the following can be used:

```
84362 command -p getconf PATH
```

84363 There are some advantages to suppressing the special characteristics of special built-ins on
84364 occasion. For example:

```
84365 command exec > unwritable-file
```

84366 does not cause a non-interactive script to abort, so that the output status can be checked by the
84367 script.

84368 The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an
84369 error occurs so that applications can distinguish “failure to find a utility” from “invoked utility
84370 exited with an error indication”. The value 127 was chosen because it is not commonly used for
84371 other meanings; most utilities use small values for “normal error conditions” and the values
84372 above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen
84373 in a similar manner to indicate that the utility could be found, but not invoked. Some scripts
84374 produce meaningful error messages differentiating the 126 and 127 cases. The distinction
84375 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
84376 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
84377 any other reason.

84378 Since the `-v` and `-V` options of *command* produce output in relation to the current shell execution
84379 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell
84380 or separate utility execution environment, such as one of the following:

```
84381 (PATH=foo command -v)
84382 nohup command -v
```

84383 it does not necessarily produce correct results. For example, when called with *nohup* or an *exec*
84384 function, in a separate utility execution environment, most implementations are not able to
84385 identify aliases, functions, or special built-ins.

84386 Two types of regular built-ins could be encountered on a system and these are described
84387 separately by *command*. The description of command search in [Section 2.9.1.1](#) (on page 2367)
84388 allows for a standard utility to be implemented as a regular built-in as long as it is found in the
84389 appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or
84390 some similar pathname. Other implementation-defined utilities that are not defined by this
84391 volume of POSIX.1-2017 might exist only as built-ins and have no pathname associated with
84392 them. These produce output identified as (regular) built-ins. Applications encountering these are
84393 not able to count on *execing* them, using them with *nohup*, overriding them with a different
84394 *PATH*, and so on.

84395 EXAMPLES

- 84396 1. Make a version of *cd* that always prints out the new working directory exactly once:

```
84397 cd() {
84398     command cd "$@" >/dev/null
84399     pwd
84400 }
```

- 84401 2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

```
84402 IFS='
84403 '
84404 # The preceding value should be <space><tab><newline>.
84405 # Set IFS to its default value.

84406 \unalias -a
84407 # Unset all possible aliases.
84408 # Note that unalias is escaped to prevent an alias
84409 # being used for unalias.

84410 unset -f command
84411 # Ensure command is not a user function.

84412 PATH="$(command -p getconf PATH):$PATH"
84413 # Put on a reliable PATH prefix.
84414 # ...
```

84415 At this point, given correct permissions on the directories called by *PATH*, the script has
84416 the ability to ensure that any utility it calls is the intended one. It is being very cautious
84417 because it assumes that implementation extensions may be present that would allow user
84418 functions to exist when it is invoked; this capability is not specified by this volume of
84419 POSIX.1-2017, but it is not prohibited as an extension. For example, the *ENV* variable
84420 precedes the invocation of the script with a user start-up script. Such a script could define
84421 functions to spoof the application.

84422 RATIONALE

84423 Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

84424 There is nothing in the description of *command* that implies the command line is parsed any
84425 differently from that of any other simple command. For example:

84426 `command a | b ; c`

84427 is not parsed in any special way that causes '|' or ';' to be treated other than a pipe operator
84428 or <semicolon> or that prevents function lookup on **b** or **c**.

84429 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since
84430 *command* also goes to the file system to search for utilities, the name *builtin* would not be
84431 intuitive.

84432 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special
84433 built-in for the following reasons:

84434 The removal of exportable functions made the special precedence of a special built-in
84435 unnecessary.

84436 A special built-in has special properties (see [Section 2.14](#), on page 2384) that were
84437 inappropriate for invoking other utilities. For example, two commands such as:

84438 `date > unwritable-file`

84439 `command date > unwritable-file`

84440 would have entirely different results; in a non-interactive script, the former would
84441 continue to execute the next command, the latter would abort. Introducing this semantic
84442 difference along with suppressing functions was seen to be non-intuitive.

84443 The `-p` option is present because it is useful to be able to ensure a safe path search that finds all
84444 the standard utilities. This search might not be identical to the one that occurs through one of the
84445 *exec* functions (as defined in the System Interfaces volume of POSIX.1-2017) when *PATH* is unset.
84446 At the very least, this feature is required to allow the script to access the correct version of *getconf*
84447 so that the value of the default path can be accurately retrieved.

84448 The *command* `-v` and `-V` options were added to satisfy requirements from users that are
84449 currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in
84450 the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what
84451 to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left
84452 unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more
84453 elaborate ¶in addition to the categories required by POSIX, it also reports on tracked aliases,
84454 exported aliases, and undefined functions.

84455 The output format of `-V` was left mostly unspecified because human users are its only audience.
84456 Applications should not be written to care about this information; they can use the output of `-v`
84457 to differentiate between various types of commands, but the additional information that may be
84458 emitted by the more verbose `-V` is not needed and should not be arbitrarily constrained in its
84459 verbosity or localization for application parsing reasons.

84460 **FUTURE DIRECTIONS**

84461 None.

84462 **SEE ALSO**

84463 [Section 2.9.1.1](#) (on page 2367), [Section 2.12](#) (on page 2381), [Section 2.14](#) (on page 2384), *sh*, *type*

84464 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

84465 XSH *exec*

84466 **CHANGE HISTORY**

84467 First released in Issue 4.

84468 **Issue 7**84469 Austin Group Interpretation 1003.1-2001 #196 is applied, changing the SYNOPSIS to allow `-p` to
84470 be used with `-v` (or `-V`).

84471 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84472 The *command* utility is moved from the User Portability Utilities option to the Base. User
84473 Portability Utilities is now an option for interactive utilities.84474 The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard
84475 `getconf_CS_PATH` with `getconf PATH`.

84476 **NAME**

84477 compress — compress data

84478 **SYNOPSIS**84479 XSI compress [-fv] [-b *bits*] [*file...*]84480 compress [-cfv] [-b *bits*] [*file*]84481 **DESCRIPTION**84482 The *compress* utility shall attempt to reduce the size of the named files by using adaptive Lempel-
84483 Ziv coding algorithm.84484 **Note:** Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,
84485 Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation.84486 Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on
84487 December 10th, 1985, and assigned to Sperry Corporation.84488 On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be
84489 changed and an error value greater than two shall be returned. Except when the output is to the
84490 standard output, each file shall be replaced by one with the extension *.Z*. If the invoking process
84491 has appropriate privileges, the ownership, modes, access time, and modification time of the
84492 original file are preserved. If appending the *.Z* to the filename would make the name exceed
84493 {NAME_MAX} bytes, the command shall fail. If no files are specified, the standard input shall be
84494 compressed to the standard output.84495 **OPTIONS**84496 The *compress* utility shall conform to XBD [Section 12.2](#) (on page 216).

84497 The following options shall be supported:

84498 **-b** *bits* Specify the maximum number of bits to use in a code. For a conforming
84499 application, the *bits* argument shall be:84500 $9 \leq bits \leq 14$ 84501 The implementation may allow *bits* values of greater than 14. The default is 14, 15,
84502 or 16.84503 **-c** Cause *compress* to write to the standard output; the input file is not changed, and
84504 no *.Z* files are created.84505 **-f** Force compression of *file*, even if it does not actually reduce the size of the file, or if
84506 the corresponding *file.Z* file already exists. If the **-f** option is not given, and the
84507 process is not running in the background, the user is prompted as to whether an
84508 existing *file.Z* file should be overwritten. If the response is affirmative, the existing
84509 file will be overwritten.84510 **-v** Write the percentage reduction of each file to standard error.84511 **OPERANDS**

84512 The following operand shall be supported:

84513 *file* A pathname of a file to be compressed.84514 **STDIN**84515 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
84516

84516 **INPUT FILES**84517 If *file* operands are specified, the input files contain the data to be compressed.84518 **ENVIRONMENT VARIABLES**84519 The following environment variables shall affect the execution of *compress*:84520 *LANG* Provide a default value for the internationalization variables that are unset or null.
84521 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
84522 variables used to determine the values of locale categories.)84523 *LC_ALL* If set to a non-empty string value, override the values of all the other
84524 internationalization variables.84525 *LC_COLLATE*84526 Determine the locale for the behavior of ranges, equivalence classes, and multi-
84527 character collating elements used in the extended regular expression defined for
84528 the **yesexpr** locale keyword in the *LC_MESSAGES* category.84529 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84530 characters (for example, single-byte as opposed to multi-byte characters in
84531 arguments), the behavior of character classes used in the extended regular
84532 expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.84533 *LC_MESSAGES*84534 Determine the locale used to process affirmative responses, and the locale used to
84535 affect the format and contents of diagnostic messages, prompts, and the output
84536 from the **-v** option written to standard error.84537 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.84538 **ASYNCHRONOUS EVENTS**

84539 Default.

84540 **STDOUT**84541 If no *file* operands are specified, or if a *file* operand is '-', or if the **-c** option is specified, the
84542 standard output contains the compressed output.84543 **STDERR**84544 The standard error shall be used only for diagnostic and prompt messages and the output from
84545 **-v**.84546 **OUTPUT FILES**84547 The output files shall contain the compressed output. The format of compressed files is
84548 unspecified and interchange of such files between implementations (including access via
84549 unspecified file sharing mechanisms) is not required by POSIX.1-2017.84550 **EXTENDED DESCRIPTION**

84551 None.

84552 **EXIT STATUS**

84553 The following exit values shall be returned:

84554 0 Successful completion.

84555 1 An error occurred.

84556 2 One or more files were not compressed because they would have increased in size (and the
84557 **-f** option was not specified).

84558 >2 An error occurred.

84559 **CONSEQUENCES OF ERRORS**

84560 The input file shall remain unmodified.

84561 **APPLICATION USAGE**

84562 The amount of compression obtained depends on the size of the input, the number of *bits* per
84563 code, and the distribution of common substrings. Typically, text such as source code or English is
84564 reduced by 50-60%. Compression is generally much better than that achieved by Huffman
84565 coding or adaptive Huffman coding (*compact*), and takes less time to compute.

84566 Although *compress* strictly follows the default actions upon receipt of a signal or when an error
84567 occurs, some unexpected results may occur. In some implementations it is likely that a partially
84568 compressed file is left in place, alongside its uncompressed input file. Since the general
84569 operation of *compress* is to delete the uncompressed file only after the *.Z* file has been
84570 successfully filled, an application should always carefully check the exit status of *compress* before
84571 arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

84572 The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the
84573 restrictions imposed by the lack of an explicit published file format). Some implementations
84574 based on 16-bit architectures cannot support 15 or 16-bit uncompression.

84575 **EXAMPLES**

84576 None.

84577 **RATIONALE**

84578 None.

84579 **FUTURE DIRECTIONS**

84580 None.

84581 **SEE ALSO**

84582 *uncompress*, *zcat*

84583 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

84584 **CHANGE HISTORY**

84585 First released in Issue 4.

84586 **Issue 6**

84587 The normative text is reworded to avoid use of the term “must” for application requirements.

84588 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

84589 **Issue 7**

84590 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84591 Austin Group Interpretation 1003.1-2001 #125 is applied, revising the ENVIRONMENT
84592 VARIABLES section.

84593 **NAME**84594 `cp` ‡copy files84595 **SYNOPSIS**84596 `cp [-Pfi] source_file target_file`84597 `cp [-Pfi] source_file... target`84598 `cp -R [-H|-L|-P] [-fi] source_file... target`84599 **DESCRIPTION**

84600 The first synopsis form is denoted by two operands, neither of which are existing files of type
 84601 directory. The `cp` utility shall copy the contents of *source_file* (or, if *source_file* is a file of type
 84602 symbolic link, the contents of the file referenced by *source_file*) to the destination path named by
 84603 *target_file*.

84604 The second synopsis form is denoted by two or more operands where the `-R` option is not
 84605 specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file
 84606 of type directory, if *target* does not exist, or if *target* does not name a directory. The `cp` utility shall
 84607 copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents of
 84608 the file referenced by *source_file*) to the destination path named by the concatenation of *target*, a
 84609 single `<slash>` character if *target* did not end in a `<slash>`, and the last component of *source_file*.

84610 The third synopsis form is denoted by two or more operands where the `-R` option is specified.
 84611 The `cp` utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination
 84612 path named as follows:

84613 If *target* exists and names an existing directory, the name of the corresponding destination
 84614 path for each file in the file hierarchy shall be the concatenation of *target*, a single `<slash>`
 84615 character if *target* did not end in a `<slash>`, and the pathname of the file relative to the
 84616 directory containing *source_file*.

84617 If *target* does not exist and two operands are specified, the name of the corresponding
 84618 destination path for *source_file* shall be *target*; the name of the corresponding destination
 84619 path for all other files in the file hierarchy shall be the concatenation of *target*, a `<slash>`
 84620 character, and the pathname of the file relative to *source_file*.

84621 It shall be an error if *target* does not exist and more than two operands are specified, or if *target*
 84622 exists and does not name a directory.

84623 In the following description, the term *dest_file* refers to the file named by the destination path.
 84624 The term *source_file* refers to the file that is being copied, whether specified as an operand or a
 84625 file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

84626 If the `-R` option was not specified, `cp` shall take actions based on the type and contents of
 84627 the file referenced by the symbolic link, and not by the symbolic link itself, unless the `-P`
 84628 option was specified.

84629 If the `-R` option was specified:

84630 ‡f none of the options `-H`, `-L`, nor `-P` were specified, it is unspecified which of `-H`,
 84631 `-L`, or `-P` will be used as a default.

84632 ‡f the `-H` option was specified, `cp` shall take actions based on the type and contents of
 84633 the file referenced by any symbolic link specified as a *source_file* operand.

84634 ‡f the `-L` option was specified, `cp` shall take actions based on the type and contents of
 84635 the file referenced by any symbolic link specified as a *source_file* operand or any
 84636 symbolic links encountered during traversal of a file hierarchy.

84637 ¶ If the `-P` option was specified, `cp` shall copy any symbolic link specified as a
 84638 `source_file` operand and any symbolic links encountered during traversal of a file
 84639 hierarchy, and shall not follow any symbolic links.

84640 For each `source_file`, the following steps shall be taken:

- 84641 1. If `source_file` references the same file as `dest_file`, `cp` may write a diagnostic message to
 84642 standard error; it shall do nothing more with `source_file` and shall go on to any remaining
 84643 files.
- 84644 2. If `source_file` is of type directory, the following steps shall be taken:
 - 84645 a. If the `-R` option was not specified, `cp` shall write a diagnostic message to standard
 84646 error, do nothing more with `source_file`, and go on to any remaining files.
 - 84647 b. If `source_file` was not specified as an operand and `source_file` is dot or dot-dot, `cp`
 84648 shall do nothing more with `source_file` and go on to any remaining files.
 - 84649 c. If `dest_file` exists and it is a file type not specified by the System Interfaces volume
 84650 of POSIX.1-2017, the behavior is implementation-defined.
 - 84651 d. If `dest_file` exists and it is not of type directory, `cp` shall write a diagnostic message
 84652 to standard error, do nothing more with `source_file` or any files below `source_file` in
 84653 the file hierarchy, and go on to any remaining files.
 - 84654 e. If the directory `dest_file` does not exist, it shall be created with file permission bits
 84655 set to the same value as those of `source_file`, modified by the file creation mask of
 84656 the user if the `-p` option was not specified, and then bitwise-inclusively OR'ed
 84657 with `S_IRWXU`. If `dest_file` cannot be created, `cp` shall write a diagnostic message to
 84658 standard error, do nothing more with `source_file`, and go on to any remaining files.
 84659 It is unspecified if `cp` attempts to copy files in the file hierarchy rooted in `source_file`.
 - 84660 f. The files in the directory `source_file` shall be copied to the directory `dest_file`, taking
 84661 the four steps (1 to 4) listed here with the files as `source_files`.
 - 84662 g. If `dest_file` was created, its file permission bits shall be changed (if necessary) to be
 84663 the same as those of `source_file`, modified by the file creation mask of the user if the
 84664 `-p` option was not specified.
 - 84665 h. The `cp` utility shall do nothing more with `source_file` and go on to any remaining
 84666 files.
- 84667 3. If `source_file` is of type regular file, the following steps shall be taken:
 - 84668 a. The behavior is unspecified if `dest_file` exists and was written by a previous step.
 84669 Otherwise, if `dest_file` exists, the following steps shall be taken:
 - 84670 i. If the `-i` option is in effect, the `cp` utility shall write a prompt to the standard
 84671 error and read a line from the standard input. If the response is not
 84672 affirmative, `cp` shall do nothing more with `source_file` and go on to any
 84673 remaining files.
 - 84674 ii. A file descriptor for `dest_file` shall be obtained by performing actions
 84675 equivalent to the `open()` function defined in the System Interfaces volume of
 84676 POSIX.1-2017 called using `dest_file` as the `path` argument, and the bitwise-
 84677 inclusive OR of `O_WRONLY` and `O_TRUNC` as the `oflag` argument.
 - 84678 iii. If the attempt to obtain a file descriptor fails and the `-f` option is in effect, `cp`
 84679 shall attempt to remove the file by performing actions equivalent to the
 84680 `unlink()` function defined in the System Interfaces volume of POSIX.1-2017

84681 called using *dest_file* as the *path* argument. If this attempt succeeds, *cp* shall
84682 continue with step 3b.

84683 b. If *dest_file* does not exist, a file descriptor shall be obtained by performing actions
84684 equivalent to the *open()* function defined in the System Interfaces volume of
84685 POSIX.1-2017 called using *dest_file* as the *path* argument, and the bitwise-inclusive
84686 OR of *O_WRONLY* and *O_CREAT* as the *oflag* argument. The file permission bits
84687 of *source_file* shall be the *mode* argument.

84688 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to
84689 standard error, do nothing more with *source_file*, and go on to any remaining files.

84690 d. The contents of *source_file* shall be written to the file descriptor. Any write errors
84691 shall cause *cp* to write a diagnostic message to standard error and continue to step
84692 3e.

84693 e. The file descriptor shall be closed.

84694 f. The *cp* utility shall do nothing more with *source_file*. If a write error occurred in
84695 step 3d, it is unspecified if *cp* continues with any remaining files. If no write error
84696 occurred in step 3d, *cp* shall go on to any remaining files.

84697 4. Otherwise, the **-R** option was specified, and the following steps shall be taken:

84698 a. The *dest_file* shall be created with the same file type as *source_file*.

84699 b. If *source_file* is a file of type FIFO, the file permission bits shall be the same as those
84700 of *source_file*, modified by the file creation mask of the user if the **-p** option was not
84701 specified. Otherwise, the permissions, owner ID, and group ID of *dest_file* are
84702 implementation-defined.

84703 If this creation fails for any reason, *cp* shall write a diagnostic message to standard
84704 error, do nothing more with *source_file*, and go on to any remaining files.

84705 c. If *source_file* is a file of type symbolic link, and the options require the symbolic link
84706 itself to be acted upon, the pathname contained in *dest_file* shall be the same as the
84707 pathname contained in *source_file*.

84708 If this fails for any reason, *cp* shall write a diagnostic message to standard error, do
84709 nothing more with *source_file*, and go on to any remaining files.

84710 If the implementation provides additional or alternate access control mechanisms (see XBD
84711 [Section 4.5](#), on page 108), their effect on copies of files is implementation-defined.

84712 OPTIONS

84713 The *cp* utility shall conform to XBD [Section 12.2](#) (on page 216).

84714 The following options shall be supported:

84715 **-f** If a file descriptor for a destination file cannot be obtained, as described in step
84716 3.a.ii., attempt to unlink the destination file and proceed.

84717 **-H** Take actions based on the type and contents of the file referenced by any symbolic
84718 link specified as a *source_file* operand.

84719 **-i** Write a prompt to standard error before copying to any existing non-directory
84720 destination file. If the response from the standard input is affirmative, the copy
84721 shall be attempted; otherwise, it shall not.

84722	-L	Take actions based on the type and contents of the file referenced by any symbolic link specified as a <i>source_file</i> operand or any symbolic links encountered during traversal of a file hierarchy.
84723		
84724		
84725	-P	Take actions on any symbolic link specified as a <i>source_file</i> operand or any symbolic link encountered during traversal of a file hierarchy.
84726		
84727	-p	Duplicate the following characteristics of each source file in the corresponding destination file:
84728		
84729		1. The time of last data modification and time of last access. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.
84730		
84731		2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.
84732		
84733		3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, <i>cp</i> shall write a diagnostic message to standard error.
84734		
84735		
84736		If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but
84737		are not duplicated in the destination file, it is unspecified whether <i>cp</i> writes a diagnostic message to standard error.
84738		
84739		
84740		The order in which the preceding characteristics are duplicated is unspecified. The <i>dest_file</i> shall not be deleted if these characteristics cannot be preserved.
84741		
84742	-R	Copy file hierarchies.
84743		Specifying more than one of the mutually-exclusive options -H , -L , and -P shall not be considered an error. The last option specified shall determine the behavior of the utility.
84744		
84745	OPERANDS	
84746		The following operands shall be supported:
84747	<i>source_file</i>	A pathname of a file to be copied. If a <i>source_file</i> operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard input.
84748		
84749	<i>target_file</i>	A pathname of an existing or nonexistent file, used for the output when a single file is copied. If a <i>target_file</i> operand is '-', it shall refer to a file named -; implementations shall not treat it as meaning standard output.
84750		
84751		
84752	<i>target</i>	A pathname of a directory to contain the copied files.
84753	STDIN	
84754		The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.
84755		
84756	INPUT FILES	
84757		The input files specified as operands may be of any file type.
84758	ENVIRONMENT VARIABLES	
84759		The following environment variables shall affect the execution of <i>cp</i> :
84760	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
84761		
84762		

84763	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
84764		
84765	<i>LC_COLLATE</i>	
84766		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category.
84767		
84768		
84769	<i>LC_CTYPE</i>	
84770		Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category.
84771		
84772		
84773		
84774	<i>LC_MESSAGES</i>	
84775		Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.
84776		
84777		
84778	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
84779	ASYNCHRONOUS EVENTS	
84780		Default.
84781	STDOUT	
84782		Not used.
84783	STDERR	
84784		A prompt shall be written to standard error under the conditions specified in the DESCRIPTION section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.
84785		
84786		
84787	OUTPUT FILES	
84788		The output files may be of any type.
84789	EXTENDED DESCRIPTION	
84790		None.
84791	EXIT STATUS	
84792		The following exit values shall be returned:
84793		0 All files were copied successfully.
84794		>0 An error occurred.
84795	CONSEQUENCES OF ERRORS	
84796		If <i>cp</i> is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.
84797		
84798		

84799 **APPLICATION USAGE**

84800 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to
84801 prevent users from creating programs that are set-user-ID or set-group-ID to them when copying
84802 files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example,
84803 if a file is set-user-ID and the copy has a different group ID than the source, a new group of users
84804 has execute permission to a set-user-ID program than did previously. In particular, this is a
84805 problem for superusers copying users' trees.

84806 **EXAMPLES**

84807 None.

84808 **RATIONALE**

84809 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally
84810 removing files when copying. Although the 4.3 BSD version does not prompt if the standard
84811 input is not a terminal, the standard developers decided that use of `-i` is a request for
84812 interaction, so when the destination path exists, the utility takes instructions from whatever
84813 responds on standard input.

84814 The exact format of the interactive prompts is unspecified. Only the general nature of the
84815 contents of prompts are specified because implementations may desire more descriptive
84816 prompts than those used on historical implementations. Therefore, an application using the `-i`
84817 option relies on the system to provide the most suitable dialog directly with the user, based on
84818 the behavior specified.

84819 The `-p` option is historical practice on BSD systems, duplicating the time of last data
84820 modification and time of last access. This volume of POSIX.1-2017 extends it to preserve the user
84821 and group IDs, as well as the file permissions. This requirement has obvious problems in that
84822 the directories are almost certainly modified after being copied. This volume of POSIX.1-2017
84823 requires that the modification times be preserved. The statement that the order in which the
84824 characteristics are duplicated is unspecified is to permit implementations to provide the
84825 maximum amount of security for the user. Implementations should take into account the
84826 obvious security issues involved in setting the owner, group, and mode in the wrong order or
84827 creating files with an owner, group, or mode different from the final value.

84828 It is unspecified whether `cp` writes diagnostic messages when the user and group IDs cannot be
84829 set due to the widespread practice of users using `-p` to duplicate some portion of the file
84830 characteristics, indifferent to the duplication of others. Historic implementations only write
84831 diagnostic messages on errors other than [EPERM].

84832 Earlier versions of this standard included support for the `-r` option to copy file hierarchies. The
84833 `-r` option is historical practice on BSD and BSD-derived systems. This option is no longer
84834 specified by POSIX.1-2017 but may be present in some implementations. The `-R` option was
84835 added as a close synonym to the `-r` option, selected for consistency with all other options in this
84836 volume of POSIX.1-2017 that do recursive directory descent.

84837 The difference between `-R` and the removed `-r` option is in the treatment by `cp` of file types other
84838 than regular and directory. It was implementation-defined how the `-` option treated special files
84839 to allow both historical implementations and those that chose to support `-r` with the same
84840 abilities as `-R` defined by this volume of POSIX.1-2017. The original `-r` flag, for historic reasons,
84841 did not handle special files any differently from regular files, but always read the file and copied
84842 its contents. This had obvious problems in the presence of special file types; for example,
84843 character devices, FIFOs, and sockets.

84844 When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy
84845 files that are on the same level in the hierarchy or above the file where the failure occurred. It is
84846 unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which

84847 cannot succeed in any case).

84848 Permissions, owners, and groups of created special file types have been deliberately left as
84849 implementation-defined. This is to allow systems to satisfy special requirements (for example,
84850 allowing users to create character special devices, but requiring them to be owned by a certain
84851 group). In general, it is strongly suggested that the permissions, owner, and group be the same
84852 as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable
84853 that additional privileges are required to create block, character, or other implementation-
84854 defined special file types.

84855 Additionally, the `-p` option explicitly requires that all set-user-ID and set-group-ID permissions
84856 be discarded if any of the owner or group IDs cannot be set. This is to keep users from
84857 unintentionally giving away special privilege when copying programs.

84858 When creating regular files, historical versions of *cp* use the mode of the source file as modified
84859 by the file mode creation mask. Other choices would have been to use the mode of the source file
84860 unmodified by the creation mask or to use the same mode as would be given to a new file
84861 created by the user (plus the execution bits of the source file) and then modify it by the file mode
84862 creation mask. In the absence of any strong reason to change historic practice, it was in large part
84863 retained.

84864 When creating directories, historical versions of *cp* use the mode of the source directory, plus
84865 read, write, and search bits for the owner, as modified by the file mode creation mask. This is
84866 done so that *cp* can copy trees where the user has read permission, but the owner does not. A
84867 side-effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the
84868 copy is done, historical versions of *cp* set the permissions on the created directory to be the same
84869 as the source directory, unmodified by the file creation mask.

84870 This behavior has been modified so that *cp* is always able to create the contents of the directory,
84871 regardless of the file creation mask. After the copy is done, the permissions are set to be the same
84872 as the source directory, as modified by the file creation mask. This latter change from historical
84873 behavior is to prevent users from accidentally creating directories with permissions beyond
84874 those they would normally set and for consistency with the behavior of *cp* in creating files.

84875 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations
84876 are strongly encouraged to do so. Historical implementations have detected the attempt in most
84877 cases.

84878 There are two methods of copying subtrees in this volume of POSIX.1-2017. The other method is
84879 described as part of the *pax* utility (see *pax*). Both methods are historical practice. The *cp* utility
84880 provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each
84881 provides additional functionality to the other; in particular, *pax* maintains the hard-link structure
84882 of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results
84883 be similar (using appropriate option combinations in both utilities). The results are not required
84884 to be identical; there seemed insufficient gain to applications to balance the difficulty of
84885 implementations having to guarantee that the results would be exactly identical.

84886 The wording allowing *cp* to copy a directory to implementation-defined file types not specified
84887 by the System Interfaces volume of POSIX.1-2017 is provided so that implementations
84888 supporting symbolic links are not required to prohibit copying directories to symbolic links.
84889 Other extensions to the System Interfaces volume of POSIX.1-2017 file types may need to use
84890 this loophole as well.

84891 **FUTURE DIRECTIONS**

84892 None.

84893 **SEE ALSO**84894 *mv, find, ln, pax*84895 XBD [Section 4.5](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)84896 XSH *open()*, *unlink()*84897 **CHANGE HISTORY**

84898 First released in Issue 2.

84899 **Issue 6**84900 The `-r` option is marked obsolescent.84901 The new options `-H`, `-L`, and `-P` are added to align with the IEEE P1003.2b draft standard. These
84902 options affect the processing of symbolic links.84903 IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the `-P` option.84904 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the
84905 SEE ALSO section.84906 **Issue 7**84907 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
84908 `LC_MESSAGES` environment variable.

84909 Austin Group Interpretations 1003.1-2001 #092, #164, #165, and #168 are applied.

84910 SD5-XCU-ERN-31 and SD5-XCU-ERN-42 are applied, updating the DESCRIPTION.

84911 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

84912 SD5-XCU-ERN-102 is applied, clarifying the `-i` option within the OPTIONS section.84913 The obsolescent `-r` option is removed.84914 The `-P` option is added to the SYNOPSIS and to the DESCRIPTION with respect to the `-R`
84915 option.

84916 **NAME**

84917 crontab — schedule periodic background work

84918 **SYNOPSIS**84919 crontab [*file*]84920 UP crontab [**-e** | **-l** | **-r**]84921 **DESCRIPTION**

84922 UP The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of
 84923 commands and the times at which they shall be executed. The new crontab entry can be input by
 84924 UP specifying *file* or input from standard input if no *file* operand is specified, or by using an editor,
 84925 if **-e** is specified.

84926 Upon execution of a command from a crontab entry, the implementation shall supply a default
 84927 environment, defining at least the following environment variables:

84928 *HOME* A pathname of the user's home directory.

84929 *LOGNAME* The user's login name.

84930 *PATH* A string representing a search path guaranteed to find all of the standard utilities.

84931 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by
 84932 this volume of POSIX.1-2017, the value shall be a pathname for *sh*.

84933 The values of these variables when *crontab* is invoked as specified by this volume of
 84934 POSIX.1-2017 shall not affect the default values provided when the scheduled command is run.

84935 If standard output and standard error are not redirected by commands executed from the
 84936 crontab entry, any generated output or errors shall be mailed, via an implementation-defined
 84937 method, to the user.

84938 XSI Users shall be permitted to use *crontab* if their names appear in the file **cron.allow** which is
 84939 located in an implementation-defined directory. If that file does not exist, the file **cron.deny**,
 84940 which is located in an implementation-defined directory, shall be checked to determine whether
 84941 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate
 84942 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage
 84943 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line.

84944 **OPTIONS**84945 The *crontab* utility shall conform to XBD [Section 12.2](#) (on page 216).

84946 The following options shall be supported:

84947 UP **-e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if
 84948 the crontab entry does not exist. When editing is complete, the entry shall be
 84949 installed as the user's crontab entry.

84950 **-l** (The letter ell.) List the invoking user's crontab entry.

84951 **-r** Remove the invoking user's crontab entry.

84952 **OPERANDS**

84953 The following operand shall be supported:

84954 *file* The pathname of a file that contains specifications, in the format defined in the
 84955 INPUT FILES section, for crontab entries.

84956 **STDIN**

84957 See the INPUT FILES section.

84958 **INPUT FILES**84959 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file
84960 consisting of lines of six fields each. The fields shall be separated by <blank> characters. The
84961 first five fields shall be integer patterns that specify the following:

- 84962 1. Minute [0,59]
- 84963 2. Hour [0,23]
- 84964 3. Day of the month [1,31]
- 84965 4. Month of the year [1,12]
- 84966 5. Day of the week ([0,6] with 0=Sunday)

84967 Each of these patterns can be either an <asterisk> (meaning all valid values), an element, or a list
84968 of elements separated by <comma> characters. An element shall be either a number or two
84969 numbers separated by a <hyphen-minus> (meaning an inclusive range). The specification of
84970 days can be made by two fields (day of the month and day of the week). If month, day of month,
84971 and day of week are all <asterisk> characters, every day shall be matched. If either the month or
84972 day of month is specified as an element or list, but the day of week is an <asterisk>, the month
84973 and day of month fields shall specify the days that match. If both month and day of month are
84974 specified as an <asterisk>, but day of week is an element or list, then only the specified days of
84975 the week match. Finally, if either the month or day of month is specified as an element or list,
84976 and the day of week is also specified as an element or list, then any day matching either the
84977 month and day of month, or the day of week, shall be matched.

84978 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified
84979 times. A <percent-sign> character in this field shall be translated to a <newline>. Any character
84980 preceded by a <backslash> (including the '%') shall cause that character to be treated literally.
84981 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the
84982 command interpreter. The other lines shall be made available to the command as standard input.

84983 Blank lines and those whose first non-<blank> is '#' shall be ignored.

84984 XSI The text files **cron.allow** and **cron.deny**, which are located in an implementation-defined
84985 directory, shall contain zero or more user names, one per line, of users who are, respectively,
84986 authorized or denied access to the service underlying the *crontab* utility.

84987 **ENVIRONMENT VARIABLES**84988 The following environment variables shall affect the execution of *crontab*:

84989 *EDITOR* Determine the editor to be invoked when the **-e** option is specified. The default
84990 editor shall be *vi*.

84991 *LANG* Provide a default value for the internationalization variables that are unset or null.
84992 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
84993 variables used to determine the values of locale categories.)

84994 *LC_ALL* If set to a non-empty string value, override the values of all the other
84995 internationalization variables.

84996 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
84997 characters (for example, single-byte as opposed to multi-byte characters in
84998 arguments and input files).

84999 *LC_MESSAGES*

85000 Determine the locale that should be used to affect the format and contents of

85001 diagnostic messages written to standard error.

85002 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

85003 **ASYNCHRONOUS EVENTS**

85004 Default.

85005 **STDOUT**

85006 If the `-l` option is specified, the crontab entry shall be written to the standard output.

85007 **STDERR**

85008 The standard error shall be used only for diagnostic messages.

85009 **OUTPUT FILES**

85010 None.

85011 **EXTENDED DESCRIPTION**

85012 None.

85013 **EXIT STATUS**

85014 The following exit values shall be returned:

85015 0 Successful completion.

85016 >0 An error occurred.

85017 **CONSEQUENCES OF ERRORS**

85018 UP The user's crontab entry is not submitted, removed, `edited`, or listed.

85019 **APPLICATION USAGE**

85020 The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other

85021 cultures may be supported with substantially different interfaces, although implementations are

85022 encouraged to provide comparable levels of functionality.

85023 The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the

85024 scheduled job are not affected by the settings of those variables when *crontab* is run; as stated,

85025 they are defaults. The text about "invoked as specified by this volume of POSIX.1-2017" means

85026 that the implementation may provide extensions that allow these variables to be affected at

85027 runtime, but that the user has to take explicit action in order to access the extension, such as give

85028 a new option flag or modify the format of the crontab entry.

85029 A typical user error is to type only *crontab*; this causes the system to wait for the new crontab

85030 entry on standard input. If end-of-file is typed (generally `<control>-D`), the crontab entry is

85031 replaced by an empty file. In this case, the user should type the interrupt character, which

85032 prevents the crontab entry from being replaced.

85033 **EXAMPLES**

85034 1. Clean up **core** files every weekday morning at 3:15 am:

85035 `15 3 * * 1-5 find "$HOME" -name core -exec rm -f {} + 2>/dev/null`

85036 2. Mail a birthday greeting:

85037 `0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.`

85038 3. As an example of specifying the two types of days:

85039 `0 0 1,15 * 1`

85040 would run a command on the first and fifteenth of each month, as well as on every
85041 Monday. To specify days by only one field, the other field should be set to '*'; for
85042 example:

```
85043 0 0 * * 1
```

85044 would run a command only on Mondays.

85045 RATIONALE

85046 All references to a *cron* daemon and to *cron files* have been omitted. Although historical
85047 implementations have used this arrangement, there is no reason to limit future implementations.

85048 This description of *crontab* is designed to support only users with normal privileges. The format
85049 of the input is based on the System V *crontab*; however, there is no requirement here that the
85050 actual system database used by the *cron* daemon (or a similar mechanism) use this format
85051 internally. For example, systems derived from BSD are likely to have an additional field
85052 appended that indicates the user identity to be used when the job is submitted.

85053 The `-e` option was adopted from the SVID as a user convenience, although it does not exist in all
85054 historical implementations.

85055 FUTURE DIRECTIONS

85056 None.

85057 SEE ALSO

85058 [*at*](#)

85059 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

85060 CHANGE HISTORY

85061 First released in Issue 2.

85062 Issue 6

85063 This utility is marked as part of the User Portability Utilities option.

85064 The normative text is reworded to avoid use of the term “must” for application requirements.

85065 Issue 7

85066 The *crontab* utility (except for the `-e` option) is moved from the User Portability Utilities option
85067 to the Base. User Portability Utilities is now an option for interactive utilities.

85068 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced
85069 by the *crontab* utility.

85070 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85071 The first example is changed to remove the unreliable use of `find | xargs`.

85072 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0079 [584] is applied.

85073 **NAME**85074 `csplit` \ddagger 'split files based on context85075 **SYNOPSIS**85076 `csplit [-ks] [-f prefix] [-n number] file arg...`85077 **DESCRIPTION**85078 The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into
85079 other files as directed by the *arg* operands, and write the sizes of the files.85080 **OPTIONS**85081 The *csplit* utility shall conform to XBD [Section 12.2](#) (on page 216).

85082 The following options shall be supported:

85083 **-f *prefix*** Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is *xx00* ... *xxn*. If
85084 the *prefix* argument would create a filename exceeding {NAME_MAX} bytes, an
85085 error shall result, *csplit* shall exit with a diagnostic message, and no files shall be
85086 created.85087 **-k** Leave previously created files intact. By default, *csplit* shall remove created files if
85088 an error occurs.85089 **-n *number*** Use *number* decimal digits to form filenames for the file pieces. The default shall be
85090 2.85091 **-s** Suppress the output of file size messages.85092 **OPERANDS**

85093 The following operands shall be supported:

85094 *file* The pathname of a text file to be split. If *file* is '-', the standard input shall be
85095 used.85096 Each *arg* operand can be one of the following:85097 */rexp/[offset]*85098 A file shall be created using the content of the lines from the current line up to, but
85099 not including, the line that results from the evaluation of the regular expression
85100 with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for
85101 basic regular expressions described in XBD [Section 9.3](#) (on page 183). The
85102 application shall use the sequence "\/" to specify a <slash> character within the
85103 *rexp*. The optional offset shall be a positive or negative integer value representing a
85104 number of lines. A positive integer value can be preceded by '+'. If the selection
85105 of lines from an *offset* expression of this type would create a file with zero lines, or
85106 one with greater than the number of lines left in the input file, the results are
85107 unspecified. After the section is created, the current line shall be set to the line that
85108 results from the evaluation of the regular expression with any offset applied. If the
85109 current line is the first line in the file and a regular expression operation has not yet
85110 been performed, the pattern match of *rexp* shall be applied from the current line to
85111 the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the
85112 line following the current line to the end of the file.85113 *%rexp%[offset]*85114 Equivalent to */rexp/[offset]*, except that no file shall be created for the selected
85115 section of the input file. The application shall use the sequence "%/" to specify a
85116 <percent-sign> character within the *rexp*.

- 85117 *line_no* Create a file from the current line up to (but not including) the line number *line_no*.
 85118 Lines in the file shall be numbered starting at one. The current line becomes
 85119 *line_no*.
- 85120 {*num*} Repeat operand. This operand can follow any of the operands described
 85121 previously. If it follows a *rexp* type operand, that operand shall be applied *num*
 85122 more times. If it follows a *line_no* operand, the file shall be split every *line_no* lines,
 85123 *num* times, from that point.
- 85124 An error shall be reported if an operand does not reference a line between the current position
 85125 and the end of the file.
- 85126 **STDIN**
- 85127 See the INPUT FILES section.
- 85128 **INPUT FILES**
- 85129 The input file shall be a text file.
- 85130 **ENVIRONMENT VARIABLES**
- 85131 The following environment variables shall affect the execution of *csplit*:
- 85132 *LANG* Provide a default value for the internationalization variables that are unset or null.
 85133 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 85134 variables used to determine the values of locale categories.)
- 85135 *LC_ALL* If set to a non-empty string value, override the values of all the other
 85136 internationalization variables.
- 85137 *LC_COLLATE*
- 85138 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 85139 character collating elements within regular expressions.
- 85140 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 85141 characters (for example, single-byte as opposed to multi-byte characters in
 85142 arguments and input files) and the behavior of character classes within regular
 85143 expressions.
- 85144 *LC_MESSAGES*
- 85145 Determine the locale that should be used to affect the format and contents of
 85146 diagnostic messages written to standard error.
- 85147 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 85148 **ASYNCHRONOUS EVENTS**
- 85149 If the **-k** option is specified, created files shall be retained. Otherwise, the default action occurs.
- 85150 **STDOUT**
- 85151 Unless the **-s** option is used, the standard output shall consist of one line per file created, with a
 85152 format as follows:
- 85153 "%d\n", <*file size in bytes*>
- 85154 **STDERR**
- 85155 The standard error shall be used only for diagnostic messages.
- 85156 **OUTPUT FILES**
- 85157 The output files shall contain portions of the original input file; otherwise, unchanged.

85158 **EXTENDED DESCRIPTION**

85159 None.

85160 **EXIT STATUS**

85161 The following exit values shall be returned:

85162 0 Successful completion.

85163 >0 An error occurred.

85164 **CONSEQUENCES OF ERRORS**85165 By default, created files shall be removed if an error occurs. When the `-k` option is specified,
85166 created files shall not be removed if an error occurs.85167 **APPLICATION USAGE**

85168 None.

85169 **EXAMPLES**85170 1. This example creates four files, `cobol00 ... cobol03`:85171 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

85172 After editing the split files, they can be recombined as follows:

85173 `cat cobol0[0-3] > file`

85174 Note that this example overwrites the original file.

85175 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up
85176 to 9999 lines; this is because lines in the file are numbered from 1 rather than zero, for
85177 historical reasons:85178 `csplit -k file 100 {99}`85179 3. Assuming that `prog.c` follows the C-language coding convention of ending routines with
85180 a `'}'` at the beginning of the line, this example creates a file containing each separate C
85181 routine (up to 21) in `prog.c`:85182 `csplit -k prog.c '%main(%' '/^}'+1' {20}`85183 **RATIONALE**85184 The `-n` option was added to extend the range of filenames that could be handled.85185 Consideration was given to adding a `-a` flag to use the alphabetic filename generation used by
85186 the historical `split` utility, but the functionality added by the `-n` option was deemed to make
85187 alphabetic naming unnecessary.85188 **FUTURE DIRECTIONS**

85189 None.

85190 **SEE ALSO**85191 *sed*, *split*85192 XBD [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on page 216)85193 **CHANGE HISTORY**

85194 First released in Issue 2.

85195 **Issue 5**

85196 The FUTURE DIRECTIONS section is added.

85197 **Issue 6**

85198 This utility is marked as part of the User Portability Utilities option.

85199 The APPLICATION USAGE section is added.

85200 The description of regular expression operands is changed to align with the IEEE P1003.2b draft
85201 standard.

85202 The normative text is reworded to avoid use of the term “must” for application requirements.

85203 **Issue 7**

85204 The *csplit* utility is moved from the User Portability Utilities option to the Base. User Portability
85205 Utilities is now an option for interactive utilities.

85206 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85207 The SYNOPSIS and OPERANDS sections are revised to use a single *arg* to split a file into two
85208 pieces.

85209 **NAME**85210 `ctags` — create a tags file (**DEVELOPMENT, FORTRAN**)85211 **SYNOPSIS**85212 SD `ctags [-a] [-f tagsfile] pathname...`85213 `ctags -x pathname...`85214 **DESCRIPTION**

85215 The `ctags` utility shall be provided on systems that support the the Software Development
 85216 Utilities option, and either or both of the C-Language Development Utilities option and
 85217 FORTRAN Development Utilities option. On other systems, it is optional.

85218 The `ctags` utility shall write a *tagsfile* or an index of objects from C-language or FORTRAN source
 85219 files specified by the *pathname* operands. The *tagsfile* shall list the locators of language-specific
 85220 objects within the source files. A locator consists of a name, *pathname*, and either a search
 85221 pattern or a line number that can be used in searching for the object definition. The objects that
 85222 shall be recognized are specified in the EXTENDED DESCRIPTION section.

85223 **OPTIONS**85224 The `ctags` utility shall conform to XBD [Section 12.2](#) (on page 216).

85225 The following options shall be supported:

85226 **-a** Append to *tagsfile*.85227 **-f** *tagsfile* Write the object locator lists into *tagsfile* instead of the default file named **tags** in the
85228 current directory.85229 **-x** Produce a list of object names, the line number, and filename in which each is
85230 defined, as well as the text of that line, and write this to the standard output. A
85231 *tagsfile* shall not be created when **-x** is specified.85232 **OPERANDS**85233 The following *pathname* operands are supported:85234 *file.c* Files with basenames ending with the **.c** suffix shall be treated as C-language
85235 source code. Such files that are not valid input to *c99* produce unspecified results.85236 *file.h* Files with basenames ending with the **.h** suffix shall be treated as C-language
85237 source code. Such files that are not valid input to *c99* produce unspecified results.85238 *file.f* Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-
85239 language source code. Such files that are not valid input to *fort77* produce
85240 unspecified results.

85241 The handling of other files is implementation-defined.

85242 **STDIN**

85243 See the INPUT FILES section.

85244 **INPUT FILES**85245 The input files shall be text files containing source code in the language indicated by the
85246 operand filename suffixes.

85247 **ENVIRONMENT VARIABLES**85248 The following environment variables shall affect the execution of *ctags*:

85249 *LANG* Provide a default value for the internationalization variables that are unset or null.
 85250 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 85251 variables used to determine the values of locale categories.)

85252 *LC_ALL* If set to a non-empty string value, override the values of all the other
 85253 internationalization variables.

85254 *LC_COLLATE*

85255 Determine the order in which output is sorted for the *-x* option. The POSIX locale
 85256 determines the order in which the *tagsfile* is written.

85257 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 85258 characters (for example, single-byte as opposed to multi-byte characters in
 85259 arguments and input files). When processing C-language source code, if the locale
 85260 is not compatible with the C locale described by the ISO C standard, the results are
 85261 unspecified.

85262 *LC_MESSAGES*

85263 Determine the locale that should be used to affect the format and contents of
 85264 diagnostic messages written to standard error.

85265 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

85266 **ASYNCHRONOUS EVENTS**

85267 Default.

85268 **STDOUT**

85269 The list of object name information produced by the *-x* option shall be written to standard
 85270 output in the following format:

85271 "%s %d %s %s", *<object-name>*, *<line-number>*, *<filename>*, *<text>*

85272 where *<text>* is the text of line *<line-number>* of file *<filename>*.

85273 **STDERR**

85274 The standard error shall be used only for diagnostic messages.

85275 **OUTPUT FILES**85276 When the *-x* option is not specified, the format of the output file shall be:

85277 "%s\t%s\t/%s/\n", *<identifier>*, *<filename>*, *<pattern>*

85278 where *<pattern>* is a search pattern that could be used by an editor to find the defining instance
 85279 of *<identifier>* in *<filename>* (where *defining instance* is indicated by the declarations listed in the
 85280 EXTENDED DESCRIPTION).

85281 An optional *<circumflex>* ('*^*') can be added as a prefix to *<pattern>*, and an optional *<dollar-sign>*
 85282 can be appended to *<pattern>* to indicate that the pattern is anchored to the beginning
 85283 (end) of a line of text. Any *<slash>* or *<backslash>* characters in *<pattern>* shall be preceded by a
 85284 *<backslash>* character. The anchoring *<circumflex>*, *<dollar-sign>*, and escaping *<backslash>*
 85285 characters shall not be considered part of the search pattern. All other characters in the search
 85286 pattern shall be considered literal characters.

85287 An alternative format is:

85288 "%s\t%s\t?%s?\n", <identifier>, <filename>, <pattern>

85289 which is identical to the first format except that <slash> characters in <pattern> shall not be
85290 preceded by escaping <backslash> characters, and <question-mark> characters in <pattern>
85291 shall be preceded by <backslash> characters.

85292 A second alternative format is:

85293 "%s\t%s\t%d\n", <identifier>, <filename>, <lineno>

85294 where <lineno> is a decimal line number that could be used by an editor to find <identifier> in
85295 <filename>.

85296 Neither alternative format shall be produced by *ctags* when it is used as described by
85297 POSIX.1-2017, but the standard utilities that process tags files shall be able to process those
85298 formats as well as the first format.

85299 In any of these formats, the file shall be sorted by identifier, based on the collation sequence in
85300 the POSIX locale.

85301 EXTENDED DESCRIPTION

85302 If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output
85303 line for each of the following objects:

85304 Function definitions

85305 Type definitions

85306 Macros with arguments

85307 It may also produce output for any of the following objects:

85308 Function prototypes

85309 Structures

85310 Unions

85311 Global variable definitions

85312 Enumeration types

85313 Macros without arguments

85314 **#define** statements

85315 **#line** statements

85316 Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C
85317 programs. The tag formed shall be created by prefixing **M** to the name of the file, with the
85318 trailing **.c**, and leading pathname components (if any) removed.

85319 On systems that do not support the C-Language Development Utilities option, *ctags* produces
85320 unspecified results for C-language source code files. It should write to standard error a message
85321 identifying this condition and cause a non-zero exit status to be produced.

85322 If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each
85323 function definition. It may also produce output for any of the following objects:

85324 Subroutine definitions

85325 COMMON statements

85326 PARAMETER statements

85327 DATA and BLOCK DATA statements

85328 Statement numbers

85329 On systems that do not support the FORTRAN Development Utilities option, *ctags* produces
 85330 unspecified results for FORTRAN source code files. It should write to standard error a message
 85331 identifying this condition and cause a non-zero exit status to be produced.

85332 It is implementation-defined what other objects (including duplicate identifiers) produce output.

85333 **EXIT STATUS**

85334 The following exit values shall be returned:

85335 0 Successful completion.

85336 >0 An error occurred.

85337 **CONSEQUENCES OF ERRORS**

85338 Default.

85339 **APPLICATION USAGE**

85340 The output with `-x` is meant to be a simple index that can be written out as an off-line readable
 85341 function index. If the input files to *ctags* (such as `.c` files) were not created using the same locale
 85342 as that in effect when *ctags* `-x` is run, results might not be as expected.

85343 The description of C-language processing says “attempts to” because the C language can be
 85344 greatly confused, especially through the use of `#defines`, and this utility would be of no use if
 85345 the real C preprocessor were run to identify them. The output from *ctags* may be fooled and
 85346 incorrect for various constructs.

85347 **EXAMPLES**

85348 None.

85349 **RATIONALE**

85350 The option list was significantly reduced from that provided by historical implementations. The
 85351 `-F` option was omitted as redundant, since it is the default. The `-B` option was omitted as being
 85352 of very limited usefulness. The `-t` option was omitted since the recognition of `typedefs` is now
 85353 required for C source files. The `-u` option was omitted because the update function was judged
 85354 to be not only inefficient, but also rarely needed.

85355 An early proposal included a `-w` option to suppress warning diagnostics. Since the types of such
 85356 diagnostics could not be described, the option was omitted as being not useful.

85357 The text for `LC_CTYPE` about compatibility with the C locale acknowledges that the ISO C
 85358 standard imposes requirements on the locale used to process C source. This could easily be a
 85359 superset of that known as “the C locale” by way of implementation extensions, or one of a few
 85360 alternative locales for systems supporting different codesets. No statement is made for
 85361 FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar
 85362 locale concept. However, a general rule in this volume of POSIX.1-2017 is that any time that
 85363 locales do not match (preparing a file for one locale and processing it in another), the results are
 85364 suspect.

85365 The collation sequence of the tags file is not affected by `LC_COLLATE` because it is typically not
 85366 used by human readers, but only by programs such as *vi* to locate the tag within the source files.
 85367 Using the POSIX locale eliminates some of the problems of coordinating locales between the
 85368 *ctags* file creator and the *vi* file reader.

85369 Historically, the tags file has been used only by *ex* and *vi*. However, the format of the tags file
 85370 has been published to encourage other programs to use the tags in new ways. The format allows
 85371 either patterns or line numbers to find the identifiers because the historical *vi* recognizes either.
 85372 The *ctags* utility does not produce the format using line numbers because it is not useful
 85373 following any source file changes that add or delete lines. The documented search patterns
 85374 match historical practice. It should be noted that literal leading `<circumflex>` or trailing `<dollar-`
 85375 `sign>` characters in the search pattern will only behave correctly if anchored to the beginning of
 85376 the line or end of the line by an additional `<circumflex>` or `<dollar-sign>` character.

85377 Historical implementations also understand the objects used by the languages Pascal and
 85378 sometimes LISP, and they understand the C source output by *lex* and *yacc*. The *ctags* utility is not
 85379 required to accommodate these languages, although implementors are encouraged to do so.

85380 The following historical option was not specified, as *vgrind* is not included in this volume of
 85381 POSIX.1-2017:

85382 `-v` If the `-v` flag is given, an index of the form expected by *vgrind* is produced on the
 85383 standard output. This listing contains the function name, filename, and page
 85384 number (assuming 64-line pages). Since the output is sorted into lexicographic
 85385 order, it may be desired to run the output through *sort -f*. Sample use:

```
85386 ctags -v files | sort -f > index vgrind -x index
```

85387 The special treatment of the tag **main** makes the use of *ctags* practical in directories with more
 85388 than one program.

85389 **FUTURE DIRECTIONS**

85390 None.

85391 **SEE ALSO**

85392 [c99](#), [fort77](#), [vi](#)

85393 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

85394 **CHANGE HISTORY**

85395 First released in Issue 4.

85396 **Issue 5**

85397 The FUTURE DIRECTIONS section is added.

85398 **Issue 6**

85399 This utility is marked as part of the User Portability Utilities option.

85400 The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

85401 The normative text is reworded to avoid use of the term “must” for application requirements.

85402 IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the
 85403 DESCRIPTION.

85404 **Issue 7**

85405 The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

85406 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85407 **NAME**85408 cut \ddagger 'cut out selected fields of each line of a file85409 **SYNOPSIS**85410 cut -b *list* [-n] [*file...*]85411 cut -c *list* [*file...*]85412 cut -f *list* [-d *delim*] [-s] [*file...*]85413 **DESCRIPTION**85414 The *cut* utility shall cut out bytes (**-b** option), characters (**-c** option), or character-delimited fields
85415 (**-f** option) from each line in one or more files, concatenate them, and write them to standard
85416 output.85417 **OPTIONS**85418 The *cut* utility shall conform to XBD [Section 12.2](#) (on page 216).85419 The application shall ensure that the option-argument *list* (see options **-b**, **-c**, and **-f** below) is a
85420 <comma>-separated list or <blank>-separated list of positive numbers and ranges. Ranges can
85421 be in three forms. The first is two positive numbers separated by a <hyphen-minus> (*low-high*),
85422 which represents all fields from the first number to the second number. The second is a positive
85423 number preceded by a <hyphen-minus> (*-high*), which represents all fields from field number 1
85424 to that number. The third is a positive number followed by a <hyphen-minus> (*low-*), which
85425 represents that number to the last field, inclusive. The elements in *list* can be repeated, can
85426 overlap, and can be specified in any order, but the bytes, characters, or fields selected shall be
85427 written in the order of the input data. If an element appears in the selection list more than once,
85428 it shall be written exactly once.

85429 The following options shall be supported:

85430 **-b list** Cut based on a *list* of bytes. Each selected byte shall be output unless the **-n** option
85431 is also specified. It shall not be an error to select bytes not present in the input line.85432 **-c list** Cut based on a *list* of characters. Each selected character shall be output. It shall
85433 not be an error to select characters not present in the input line.85434 **-d delim** Set the field delimiter to the character *delim*. The default is the <tab>.85435 **-f list** Cut based on a *list* of fields, assumed to be separated in the file by a delimiter
85436 character (see **-d**). Each selected field shall be output. Output fields shall be
85437 separated by a single occurrence of the field delimiter character. Lines with no field
85438 delimiters shall be passed through intact, unless **-s** is specified. It shall not be an
85439 error to select fields not present in the input line.85440 **-n** Do not split characters. When specified with the **-b** option, each element in *list* of
85441 the form *low-high* (<hyphen-minus>-separated numbers) shall be modified as
85442 follows:85443 If the byte selected by *low* is not the first byte of a character, *low* shall be
85444 decremented to select the first byte of the character originally selected by *low*.
85445 If the byte selected by *high* is not the last byte of a character, *high* shall be
85446 decremented to select the last byte of the character prior to the character
85447 originally selected by *high*, or zero if there is no prior character. If the
85448 resulting range element has *high* equal to zero or *low* greater than *high*, the list
85449 element shall be dropped from *list* for that input line without causing an
85450 error.85451 Each element in *list* of the form *low-* shall be treated as above with *high* set to the

85452 number of bytes in the current line, not including the terminating <newline>. Each
 85453 element in *list* of the form *-high* shall be treated as above with *low* set to 1. Each
 85454 element in *list* of the form *num* (a single number) shall be treated as above with *low*
 85455 set to *num* and *high* set to *num*.

85456 **-s** Suppress lines with no delimiter characters, when used with the **-f** option. Unless
 85457 specified, lines with no delimiters shall be passed through untouched.

85458 OPERANDS

85459 The following operand shall be supported:

85460 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
 85461 '-', the standard input shall be used.

85462 STDIN

85463 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
 85464 See the INPUT FILES section.

85465 INPUT FILES

85466 The input files shall be text files, except that line lengths shall be unlimited.

85467 ENVIRONMENT VARIABLES

85468 The following environment variables shall affect the execution of *cut*:

85469 *LANG* Provide a default value for the internationalization variables that are unset or null.
 85470 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 85471 variables used to determine the values of locale categories.)

85472 *LC_ALL* If set to a non-empty string value, override the values of all the other
 85473 internationalization variables.

85474 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 85475 characters (for example, single-byte as opposed to multi-byte characters in
 85476 arguments and input files).

85477 *LC_MESSAGES*

85478 Determine the locale that should be used to affect the format and contents of
 85479 diagnostic messages written to standard error.

85480 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

85481 ASYNCHRONOUS EVENTS

85482 Default.

85483 STDOUT

85484 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of
 85485 the following):

85486 "%s\n", <concatenation of bytes>

85487 "%s\n", <concatenation of characters>

85488 "%s\n", <concatenation of fields and field delimiters>

85489 STDERR

85490 The standard error shall be used only for diagnostic messages.

85491 **OUTPUT FILES**

85492 None.

85493 **EXTENDED DESCRIPTION**

85494 None.

85495 **EXIT STATUS**

85496 The following exit values shall be returned:

85497 0 All input files were output successfully.

85498 >0 An error occurred.

85499 **CONSEQUENCES OF ERRORS**

85500 Default.

85501 **APPLICATION USAGE**

85502 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.
 85503 The *cut* utility should be used when the number of lines (or records) needs to remain constant.
 85504 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

85505 Earlier versions of the *cut* utility worked in an environment where bytes and characters were
 85506 considered equivalent (modulo <backspace> and <tab> processing in some implementations). In
 85507 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option
 85508 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The
 85509 algorithm specified for **-n** guarantees that:

85510 `cut -b 1-500 -n file > file1`85511 `cut -b 501- -n file > file2`

85512 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,
 85513 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

85514 **EXAMPLES**

85515 Examples of the option qualifier list:

85516 `1,4,7` Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.85517 `1-3,8` Equivalent to `1,2,3,8`.85518 `-5,10` Equivalent to `1,2,3,4,5,10`.85519 `3-` Equivalent to third to last, inclusive.

85520 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte
 85521 characters; see the description of **-n**.

85522 The following command:

85523 `cut -d : -f 1,6 /etc/passwd`

85524 reads the System V password file (user database) and produces lines of the form:

85525 `<user ID>:<home directory>`

85526 Most utilities in this volume of POSIX.1-2017 work on text files. The *cut* utility can be used to
 85527 turn files with arbitrary line lengths into a set of text files containing the same data. The *paste*
 85528 utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**
 85529 contains long lines:

85530 `cut -b 1-500 -n file > file1`85531 `cut -b 501- -n file > file2`

85532 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that
 85533 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in
 85534 **file** that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from **file1**
 85535 and **file2** using the command:

```
85536 paste -d "\0" file1 file2 > file
```

85537 RATIONALE

85538 Some historical implementations do not count <backspace> characters in determining character
 85539 counts with the `-c` option. This may be useful for using `cut` for processing `nroff` output. It was
 85540 deliberately decided not to have the `-c` option treat either <backspace> or <tab> characters in
 85541 any special fashion. The `fold` utility does treat these characters specially.

85542 Unlike other utilities, some historical implementations of `cut` exit after not finding an input file,
 85543 rather than continuing to process the remaining `file` operands. This behavior is prohibited by this
 85544 volume of POSIX.1-2017, where only the exit status is affected by this problem.

85545 The behavior of `cut` when provided with either mutually-exclusive options or options that do
 85546 not work logically together has been deliberately left unspecified in favor of global wording in
 85547 [Section 1.4](#) (on page 2336).

85548 The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The
 85549 change represents historical practice on all known systems. The original standard was
 85550 ambiguous on the nature of the output.

85551 The `list` option-arguments are historically used to select the portions of the line to be written, but
 85552 do not affect the order of the data. For example:

```
85553 echo abcdefghi | cut -c6,2,4-7,1
```

85554 yields "abdefg".

85555 A proposal to enhance `cut` with the following option:

85556 `-o` Preserve the selected field order. When this option is specified, each byte, character, or field
 85557 (or ranges of such) shall be written in the order specified by the `list` option-argument, even if
 85558 this requires multiple outputs of the same bytes, characters, or fields.

85559 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft
 85560 standard.

85561 FUTURE DIRECTIONS

85562 None.

85563 SEE ALSO

85564 [Section 2.5](#) (on page 2349), [fold](#), [grep](#), [paste](#)

85565 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

85566 CHANGE HISTORY

85567 First released in Issue 2.

85568 Issue 6

85569 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

85570 The normative text is reworded to avoid use of the term “must” for application requirements.

85571 **Issue 7**

85572 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85573 SD5-XCU-ERN-171 is applied, adding APPLICATION USAGE.

85574 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0080 [584] is applied.

85575 **NAME**85576 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)85577 **SYNOPSIS**

```
85578 XSI  cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...
85579      [-U name]... file...
```

85580 **DESCRIPTION**

85581 The *cxref* utility shall analyze a collection of C-language *files* and attempt to build a cross-
 85582 reference table. Information from **#define** lines shall be included in the symbol table. A sorted
 85583 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*
 85584 separately, or with the **-c** option, in combination. Each symbol shall contain an <asterisk> before
 85585 the declaring reference.

85586 **OPTIONS**

85587 The *cxref* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the order of the **-D**,
 85588 **-I**, and **-U** options (which are identical to their interpretation by *c99*) is significant. The
 85589 following options shall be supported:

- 85590 **-c** Write a combined cross-reference of all input files.
- 85591 **-s** Operate silently; do not print input filenames.
- 85592 **-o file** Direct output to named *file*.
- 85593 **-w num** Format output no wider than *num* (decimal) columns. This option defaults to 80 if
 85594 *num* is not specified or is less than 51.
- 85595 **-D** Equivalent to *c99*.
- 85596 **-I** Equivalent to *c99*.
- 85597 **-U** Equivalent to *c99*.

85598 **OPERANDS**

85599 The following operand shall be supported:

- 85600 *file* A pathname of a C-language source file.

85601 **STDIN**

85602 Not used.

85603 **INPUT FILES**

85604 The input files are C-language source files.

85605 **ENVIRONMENT VARIABLES**

85606 The following environment variables shall affect the execution of *cxref*:

- 85607 **LANG** Provide a default value for the internationalization variables that are unset or null.
 85608 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 85609 variables used to determine the values of locale categories.)
- 85610 **LC_ALL** If set to a non-empty string value, override the values of all the other
 85611 internationalization variables.
- 85612 **LC_COLLATE**
 85613 Determine the locale for the ordering of the output.
- 85614 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 85615 characters (for example, single-byte as opposed to multi-byte characters in
 85616 arguments and input files).

- 85617 *LC_MESSAGES*
85618 Determine the locale that should be used to affect the format and contents of
85619 diagnostic messages written to standard error.
- 85620 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 85621 **ASYNCHRONOUS EVENTS**
85622 Default.
- 85623 **STDOUT**
85624 The standard output shall be used for the cross-reference listing, unless the **-o** option is used to
85625 select a different output file.
- 85626 The format of standard output is unspecified, except that the following information shall be
85627 included:
- 85628 If the **-c** option is not specified, each portion of the listing shall start with the name of the
85629 input file on a separate line.
- 85630 The name line shall be followed by a sorted list of symbols, each with its associated
85631 location pathname, the name of the function in which it appears (if it is not a function
85632 name itself), and line number references.
- 85633 Each line number may be preceded by an <asterisk> ('*') flag, meaning that this is the
85634 declaring reference. Other single-character flags, with implementation-defined meanings,
85635 may be included.
- 85636 **STDERR**
85637 The standard error shall be used only for diagnostic messages.
- 85638 **OUTPUT FILES**
85639 The output file named by the **-o** option shall be used instead of standard output.
- 85640 **EXTENDED DESCRIPTION**
85641 None.
- 85642 **EXIT STATUS**
85643 The following exit values shall be returned:
- 85644 0 Successful completion.
85645 >0 An error occurred.
- 85646 **CONSEQUENCES OF ERRORS**
85647 Default.
- 85648 **APPLICATION USAGE**
85649 None.
- 85650 **EXAMPLES**
85651 None.
- 85652 **RATIONALE**
85653 None.
- 85654 **FUTURE DIRECTIONS**
85655 None.

85656 **SEE ALSO**85657 [c99](#)85658 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)85659 **CHANGE HISTORY**

85660 First released in Issue 2.

85661 **Issue 5**85662 In the SYNOPSIS, [-U *dir*] is changed to [-U *name*].85663 **Issue 6**

85664 The APPLICATION USAGE section is added.

85665 **Issue 7**

85666 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

85667 **NAME**

85668 date ‡write the date and time

85669 **SYNOPSIS**

85670 date [-u] [+format]

85671 XSI date [-u] *mmddhhmm*[[*cc*]*yy*]85672 **DESCRIPTION**

85673 XSI The *date* utility shall write the date and time to standard output or attempt to set the system
 85674 **date and time.** By default, the current date and time shall be written. If an operand beginning
 85675 with '+' is specified, the output format of *date* shall be controlled by the conversion
 85676 specifications and other text in the operand.

85677 **OPTIONS**85678 The *date* utility shall conform to XBD [Section 12.2](#) (on page 216).

85679 The following option shall be supported:

85680 **-u** Perform operations as if the *TZ* environment variable was set to the string "UTC0",
 85681 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the timezone
 85682 indicated by the *TZ* environment variable or the system default if that variable is
 85683 unset or null.

85684 **OPERANDS**

85685 The following operands shall be supported:

85686 **+format** When the format is specified, each conversion specifier shall be replaced in the
 85687 standard output by its corresponding value. All other characters shall be copied to
 85688 the output without change. The output shall always be terminated with a
 85689 <newline>.

85690 **Conversion Specifications**

85691 **%a** Locale's abbreviated weekday name.

85692 **%A** Locale's full weekday name.

85693 **%b** Locale's abbreviated month name.

85694 **%B** Locale's full month name.

85695 **%c** Locale's appropriate date and time representation.

85696 **%C** Century (a year divided by 100 and truncated to an integer) as a decimal
 85697 number [00,99].

85698 **%d** Day of the month as a decimal number [01,31].

85699 **%D** Date in the format *mm/dd/yy*.

85700 **%e** Day of the month as a decimal number [1,31] in a two-digit field with
 85701 leading <space> character fill.

85702 **%h** A synonym for %b.

85703 **%H** Hour (24-hour clock) as a decimal number [00,23].

85704 **%I** Hour (12-hour clock) as a decimal number [01,12].

85705	%j	Day of the year as a decimal number [001,366].
85706	%m	Month as a decimal number [01,12].
85707	%M	Minute as a decimal number [00,59].
85708	%n	A <newline>.
85709	%p	Locale's equivalent of either AM or PM.
85710	%r	12-hour clock time [01,12] using the AM/PM notation; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
85711		
85712	%S	Seconds as a decimal number [00,60].
85713	%t	A <tab>.
85714	%T	24-hour clock time [00,23] in the format <i>HH:MM:SS</i> .
85715	%u	Weekday as a decimal number [1,7] (1=Monday).
85716	%U	Week of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.
85717		
85718		
85719	%V	Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.
85720		
85721		
85722		
85723	%w	Weekday as a decimal number [0,6] (0=Sunday).
85724	%W	Week of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday shall be considered to be in week 0.
85725		
85726		
85727	%x	Locale's appropriate date representation.
85728	%X	Locale's appropriate time representation.
85729	%y	Year within century [00,99].
85730	%Y	Year with century as a decimal number.
85731	%Z	Timezone name, or no characters if no timezone is determinable.
85732	%%	A <percent-sign> character.

85733 See XBD [Section 7.3.5](#) (on page 159) for the conversion specifier values in the
 85734 POSIX locale.

85735 **Modified Conversion Specifications**

85736 Some conversion specifiers can be modified by the `E` and `O` modifier characters to
 85737 indicate a different format or specification as specified in the `LC_TIME` locale
 85738 description (see XBD [Section 7.3.5](#), on page 159). If the corresponding keyword
 85739 (see `era`, `era_year`, `era_d_fmt`, and `alt_digits` in XBD [Section 7.3.5](#), on page 159) is
 85740 not specified or not supported for the current locale, the unmodified conversion
 85741 specifier value shall be used.

85742 %Ec Locale's alternative appropriate date and time representation.

85743	%EC	The name of the base year (period) in the locale's alternative representation.
85744		
85745	%Ex	Locale's alternative date representation.
85746	%EX	Locale's alternative time representation.
85747	%Ey	Offset from %EC (year only) in the locale's alternative representation.
85748	%EY	Full alternative year representation.
85749	%Od	Day of month using the locale's alternative numeric symbols.
85750	%Oe	Day of month using the locale's alternative numeric symbols.
85751	%OH	Hour (24-hour clock) using the locale's alternative numeric symbols.
85752	%OI	Hour (12-hour clock) using the locale's alternative numeric symbols.
85753	%Om	Month using the locale's alternative numeric symbols.
85754	%OM	Minutes using the locale's alternative numeric symbols.
85755	%OS	Seconds using the locale's alternative numeric symbols.
85756	%Ou	Weekday as a number in the locale's alternative representation (Monday = 1).
85757		
85758	%OU	Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
85759		
85760	%OV	Week number of the year (Monday as the first day of the week, rules corresponding to %V), using the locale's alternative numeric symbols.
85761		
85762	%Ow	Weekday as a number in the locale's alternative representation (Sunday = 0).
85763		
85764	%OW	Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
85765		
85766	%Oy	Year (offset from %C) in alternative representation.
85767	XSI	<code>mmdhhmm[[cc]yy]</code>
85768		Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the minute (number); <i>cc</i> is the century and is the first two digits of the year (this is optional); <i>yy</i> is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if <i>yy</i> is omitted.
85769		
85770		
85771		
85772		
85773		
85774		
85775		
85776		
85777	Note:	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
85778		
85779		
85780	STDIN	
85781		Not used.

85782 **INPUT FILES**

85783 None.

85784 **ENVIRONMENT VARIABLES**85785 The following environment variables shall affect the execution of *date*:

85786 *LANG* Provide a default value for the internationalization variables that are unset or null.
 85787 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 85788 variables used to determine the values of locale categories.)

85789 *LC_ALL* If set to a non-empty string value, override the values of all the other
 85790 internationalization variables.

85791 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 85792 characters (for example, single-byte as opposed to multi-byte characters in
 85793 arguments).

85794 *LC_MESSAGES*

85795 Determine the locale that should be used to affect the format and contents of
 85796 diagnostic messages written to standard error.

85797 *LC_TIME* Determine the format and contents of date and time strings written by *date*.

85798 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

85799 *TZ* Determine the timezone in which the time and date are written, unless the *-u*
 85800 option is specified. If the *TZ* variable is unset or null and *-u* is not specified, an
 85801 unspecified system default timezone is used.

85802 **ASYNCHRONOUS EVENTS**

85803 Default.

85804 **STDOUT**

85805 When no formatting operand is specified, the output in the POSIX locale shall be equivalent to
 85806 specifying:

85807 *date* "+%a %b %e %H:%M:%S %Z %Y"

85808 **STDERR**

85809 The standard error shall be used only for diagnostic messages.

85810 **OUTPUT FILES**

85811 None.

85812 **EXTENDED DESCRIPTION**

85813 None.

85814 **EXIT STATUS**

85815 The following exit values shall be returned:

85816 0 The date was written successfully.

85817 >0 An error occurred.

85818 **CONSEQUENCES OF ERRORS**

85819 Default.

85820 **APPLICATION USAGE**

85821 Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can
 85822 contain <newline> characters in some locales, so it may be difficult to use the format shown in
 85823 standard output for parsing the output of *date* in those locales.

85824 The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap
 85825 second.

85826 Although certain of the conversion specifiers in the POSIX locale (such as the name of the
 85827 month) are shown with initial capital letters, this need not be the case in other locales. Programs
 85828 using these fields may need to adjust the capitalization if the output is going to be used at the
 85829 beginning of a sentence.

85830 The date string formatting capabilities are intended for use in Gregorian-style calendars,
 85831 possibly with a different starting year (or years). The %x and %c conversion specifications,
 85832 however, are intended for local representation; these may be based on a different, non-Gregorian
 85833 calendar.

85834 The %C conversion specification was introduced to allow a fallback for the %EC (alternative year
 85835 format base year); it can be viewed as the base of the current subdivision in the Gregorian
 85836 calendar. The century number is calculated as the year divided by 100 and truncated to an
 85837 integer; it should not be confused with the use of ordinal numbers for centuries (for example,
 85838 "twenty-first century".) Both the %EY and %Y can then be viewed as the offset from %EC and %C,
 85839 respectively.

85840 The E and O modifiers modify the traditional conversion specifiers, so that they can always be
 85841 used, even if the implementation (or the current locale) does not support the modifier.

85842 The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as
 85843 these are based on the Gregorian calendar system. Extending the E modifiers to other date
 85844 elements may provide an implementation-defined extension capable of supporting other
 85845 calendar systems, especially in combination with the O modifier.

85846 The O modifier supports time and date formats using the locale's alternative numerical symbols,
 85847 such as Kanji or Hindi digits or ordinal number representation.

85848 Non-European locales, whether they use Latin digits in computational items or not, often have
 85849 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a
 85850 variant of dates uses Roman numerals for the months: the third day of September 1991 would be
 85851 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking
 85852 countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %Y conversion
 85853 specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O
 85854 modifier was introduced to support the use for display purposes of non-Latin digits. In the
 85855 LC_TIME category in *localedef*, the optional **alt_digits** keyword is intended for this purpose. As
 85856 an example, assume the following (partial) *localedef* source:

```
85857 alt_digits  " ";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \  

85858            "IX";"X";"XI";"XII"  

85859 d_fmt      "%e.%Om.%Y"
```

85860 With the above date, the command:

```
85861 date "+%x"
```

85862 would yield 3.IX.1991. With the same **d_fmt**, but without the **alt_digits**, the command would
 85863 yield 3.9.1991.

85864 **EXAMPLES**

85865 1. The following are input/output examples of *date* used at arbitrary times in the POSIX
85866 locale:

```
85867 $ date
85868 Tue Jun 26 09:58:10 PDT 1990

85869 $ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
85870 DATE: 11/02/91
85871 TIME: 13:36:16
```

```
85872 $ date "+TIME: %r"
85873 TIME: 01:36:32 PM
```

85874 2. Examples for Denmark, where the default date and time format is %a %d %b %Y %T %Z:

```
85875 $ LANG=da_DK.iso_8859-1 date
85876 ons 02 okt 1991 15:03:32 CET

85877 $ LANG=da_DK.iso_8859-1 \
85878 date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
85879 DATO: onsdag den 2. oktober 1991
85880 KLOKKEN: 15:03:56
```

85881 3. Examples for Germany, where the default date and time format is %a %d.%h.%Y, %T %Z:

```
85882 $ LANG=De_DE.88591 date
85883 Mi 02.Okt.1991, 15:01:21 MEZ

85884 $ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
85885 DATUM: Mittwoch, 02. Oktober 1991
85886 ZEIT: 15:02:02
```

85887 4. Examples for France, where the default date and time format is %a %d %h %Y %Z %T:

```
85888 $ LANG=Fr_FR.88591 date
85889 Mer 02 oct 1991 MET 15:03:32

85890 $ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
85891 JOUR: Mercredi 02 octobre 1991
85892 HEURE: 15:03:56
```

85893 **RATIONALE**

85894 Some of the new options for formatting are from the ISO C standard. The `-u` option was
85895 introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is
85896 allowed as an equivalent *TZ* value to be compatible with all of the systems using the BSD
85897 implementation, where this option originated.

85898 The %e format conversion specification (adopted from System V) was added because the ISO C
85899 standard conversion specifications did not provide any way to produce the historical default
85900 *date* output during the first nine days of any month.

85901 There are two varieties of day and week numbering supported (in addition to any others created
85902 with the locale-dependent %E and %O modifier characters):

85903 The historical variety in which Sunday is the first day of the week and the weekdays
85904 preceding the first Sunday of the year are considered week 0. These are represented by %w
85905 and %U. A variant of this is %W, using Monday as the first day of the week, but still
85906 referring to week 0. This view of the calendar was retained because so many historical

85907 applications depend on it and the ISO C standard *strptime()* function, on which many *date*
85908 implementations are based, was defined in this way.

85909 The international standard, based on the ISO 8601:2004 standard where Monday is the first
85910 weekday and the algorithm for the first week number is more complex: If the week
85911 (Monday to Sunday) containing January 1 has four or more days in the new year, then it is
85912 week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These
85913 are represented by the new conversion specifications *%u* and *%V*, added as a result of
85914 international comments.

85915 **FUTURE DIRECTIONS**

85916 None.

85917 **SEE ALSO**

85918 XBD [Section 7.3.5](#) (on page 159), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

85919 XSH [fprintf\(\)](#), [strptime\(\)](#)

85920 **CHANGE HISTORY**

85921 First released in Issue 2.

85922 **Issue 5**

85923 Changes are made for Year 2000 alignment.

85924 **Issue 6**

85925 The following new requirements on POSIX implementations derive from alignment with the
85926 Single UNIX Specification:

85927 The *%EX* modified conversion specification is added.

85928 The Open Group Corrigendum U048/2 is applied, correcting the examples.

85929 The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors
85930 for consistency with the *LC_TIME* category.

85931 A clarification is made such that the current year is the default if the *yy* argument is omitted
85932 when setting the system date and time.

85933 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE
85934 HISTORY section.

85935 **NAME**

85936 dd ‡convert and copy a file

85937 **SYNOPSIS**85938 dd [*operand...*]85939 **DESCRIPTION**

85940 The *dd* utility shall copy the specified input file to the specified output file with possible
 85941 conversions using specific input and output block sizes. It shall read the input one block at a
 85942 time, using the specified input block size; it shall then process the block of data actually
 85943 returned, which could be smaller than the requested block size. It shall apply any conversions
 85944 that have been specified and write the resulting data to the output in blocks of the specified
 85945 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,
 85946 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a
 85947 separate output block; if the read returns less than a full block and the **sync** conversion is not
 85948 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**
 85949 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the
 85950 input shall be processed and collected into full-sized output blocks until the end of the input is
 85951 reached.

85952 The processing order shall be as follows:

- 85953 1. An input block is read.
- 85954 2. If the input block is shorter than the specified input block size and the **sync** conversion is
 85955 specified, null bytes shall be appended to the input data up to the specified size. (If either
 85956 **block** or **unblock** is also specified, <space> characters shall be appended instead of null
 85957 bytes.) The remaining conversions and output shall include the pad characters as if they
 85958 had been read from the input.
- 85959 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is
 85960 requested, the resulting data shall be written to the output as a single block, and the
 85961 remaining steps are omitted.
- 85962 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If
 85963 there is an odd number of bytes in the input block, the last byte in the input record shall
 85964 not be swapped.
- 85965 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These
 85966 conversions shall operate on the input data independently of the input blocking; an input
 85967 or output fixed-length record may span block boundaries.
- 85968 6. The data resulting from input or conversion or both shall be aggregated into output
 85969 blocks of the specified size. After the end of input is reached, any remaining output shall
 85970 be written as a block without padding if **conv=sync** is not specified; thus, the final output
 85971 block may be shorter than the output block size.

85972 **OPTIONS**

85973 None.

85974 **OPERANDS**85975 All of the operands shall be processed before any input is read. The following operands shall be
85976 supported:

- 85977 **if=file** Specify the input pathname; the default is standard input.
- 85978 **of=file** Specify the output pathname; the default is standard output. If the **seek=expr**
 85979 conversion is not also specified, the output file shall be truncated before the copy
 85980 begins if an explicit **of=file** operand is specified, unless **conv=notrunc** is specified.

85981		If seek=expr is specified, but conv=notrunc is not, the effect of the copy shall be to preserve the blocks in the output file over which <i>dd</i> seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy. If the input file is empty and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist, the size of the output file shall be set to the file offset after the seek.)
85982		
85983		
85984		
85985		
85986		
85987		
85988		
85989	ibs=expr	Specify the input block size, in bytes, by <i>expr</i> (default is 512).
85990	obs=expr	Specify the output block size, in bytes, by <i>expr</i> (default is 512).
85991	bs=expr	Set both input and output block sizes to <i>expr</i> bytes, superseding ibs= and obs= . If no conversion other than sync , noerror , and notrunc is specified, each input block shall be copied to the output as a single block without aggregating short blocks.
85992		
85993		
85994	cbs=expr	Specify the conversion block size for block and unblock in bytes by <i>expr</i> (default is zero). If cbs= is omitted or given a value of zero, using block or unblock produces unspecified results.
85995		
85996		
85997	XSI	The application shall ensure that this operand is also specified if the conv= operand is specified with a value of ascii , ebcdic , or ibm . For a conv= operand with an ascii value, the input is handled as described for the unblock value, except that characters are converted to ASCII before any trailing <space> characters are deleted. For conv= operands with ebcdic or ibm values, the input is handled as described for the block value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space> characters are added.
85998		
85999		
86000		
86001		
86002		
86003		
86004	skip=n	Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.
86005		
86006		
86007	seek=n	Skip <i>n</i> blocks (using the specified output block size) from the beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.
86008		
86009		
86010		
86011		
86012	count=n	Copy only <i>n</i> input blocks. If <i>n</i> is zero, it is unspecified whether no blocks or all blocks are copied.
86013		
86014	conv=value[,value ...]	
86015		Where <i>values</i> are <comma>-separated symbols from the following list:
86016	XSI	ascii Convert EBCDIC to ASCII; see Table 4-7 (on page 2644).
86017	XSI	ebcdic Convert ASCII to EBCDIC; see Table 4-7 (on page 2644).
86018	XSI	ibm Convert ASCII to a different EBCDIC set; see Table 4-8 (on page 2645).
86019	XSI	The ascii , ebcdic , and ibm values are mutually-exclusive.
86020	block	Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space> characters shall be appended to lines that are shorter than their conversion block size to fill the block.
86021		
86022		
86023		
86024		
86025		

86026 Lines that are longer than the conversion block size shall be truncated
 86027 to the largest number of characters that fit into that size; the number of
 86028 truncated lines shall be reported (see the STDERR section).

86029 The **block** and **unblock** values are mutually-exclusive.

86030 **unblock** Convert fixed-length records to variable length. Read a number of bytes
 86031 equal to the conversion block size (or the number of bytes remaining in
 86032 the input, if less than the conversion block size), delete all trailing
 86033 <space> characters, and append a <newline>.

86034 **lcase** Map uppercase characters specified by the *LC_CTYPE* keyword
 86035 **tolower** to the corresponding lowercase character. Characters for which
 86036 no mapping is specified shall not be modified by this conversion.

86037 The **lcase** and **ucase** symbols are mutually-exclusive.

86038 **ucase** Map lowercase characters specified by the *LC_CTYPE* keyword
 86039 **toupper** to the corresponding uppercase character. Characters for
 86040 which no mapping is specified shall not be modified by this conversion.

86041 **swab** Swap every pair of input bytes.

86042 **noerror** Do not stop processing on an input error. When an input error occurs, a
 86043 diagnostic message shall be written on standard error, followed by the
 86044 current input and output block counts in the same format as used at
 86045 completion (see the STDERR section). If the **sync** conversion is
 86046 specified, the missing input shall be replaced with null bytes and
 86047 processed normally; otherwise, the input block shall be omitted from
 86048 the output.

86049 **notrunc** Do not truncate the output file. Preserve blocks in the output file not
 86050 explicitly written by this invocation of the *dd* utility. (See also the
 86051 preceding **of=file** operand.)

86052 **sync** Pad every input block to the size of the **ibs=** buffer, appending null
 86053 bytes. (If either **block** or **unblock** is also specified, append <space>
 86054 characters, rather than null bytes.)

86055 The behavior is unspecified if operands other than **conv=** are specified more than once.

86056 For the **bs=**, **cbs=**, **ibs=**, and **obs=** operands, the application shall supply an expression
 86057 specifying a size in bytes. The expression, *expr*, can be:

- 86058 1. A positive decimal number
- 86059 2. A positive decimal number followed by *k*, specifying multiplication by 1 024
- 86060 3. A positive decimal number followed by *b*, specifying multiplication by 512
- 86061 4. Two or more positive decimal numbers (with or without *k* or *b*) separated by *x*, specifying
 86062 the product of the indicated values

86063 All of the operands are processed before any input is read.

86064 XSI The following two tables display the octal number character values used for the **ascii** and **ebcdic**
 86065 conversions (first table) and for the **ibm** conversion (second table). In both tables, the ASCII
 86066 values are the row and column headers and the EBCDIC values are found at their intersections.
 86067 For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The
 86068 inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one

86069
86070

correspondence with these tables. The differences between the two tables are highlighted by small boxes drawn around five entries.

86071

Table 4-7 ASCII to EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 .
0050	0115 (0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [0340 \	0275]	0232	0155 -
0140	0171 ,	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117	0320 }	0137 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0152 i	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0112 ¢	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0241	0276	0277
0350	0312	0313	0314 J	0315	0316 y	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 r	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377 EO

Table 4-8 ASCII to IBM EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 .
0050	0115 (0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [0340 \	0275]	0137 ^	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117	0320 }	0241 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0232	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0255 [0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0275]	0276	0277
0350	0312	0313	0314 J	0315	0316 y	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 r	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377 EO

86073 **STDIN**
 86074 If no **if=** operand is specified, the standard input shall be used. See the INPUT FILES section.

86075 **INPUT FILES**
 86076 The input file can be any file type.

86077 **ENVIRONMENT VARIABLES**
 86078 The following environment variables shall affect the execution of *dd*:

86079 **LANG** Provide a default value for the internationalization variables that are unset or null.
 86080 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 86081 variables used to determine the values of locale categories.)

86082 **LC_ALL** If set to a non-empty string value, override the values of all the other
 86083 internationalization variables.

86084 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 86085 characters (for example, single-byte as opposed to multi-byte characters in
 86086 arguments and input files), the classification of characters as uppercase or
 86087 lowercase, and the mapping of characters from one case to the other.

86088 **LC_MESSAGES**
 86089 Determine the locale that should be used to affect the format and contents of
 86090 diagnostic messages written to standard error and informative messages written to
 86091 standard output.

86092 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

86093 **ASYNCHRONOUS EVENTS**
 86094 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to
 86095 standard error, and exit as though terminated by SIGINT. It shall take the standard action for all
 86096 other signals; see the ASYNCHRONOUS EVENTS section in [Section 1.4](#) (on page 2336).

86097 **STDOUT**
 86098 If no **of=** operand is specified, the standard output shall be used. The nature of the output
 86099 depends on the operands selected.

86100 **STDERR**
 86101 On completion, *dd* shall write the number of input and output blocks to standard error. In the
 86102 POSIX locale the following formats shall be used:

86103 "%u+%u records in\n", <number of whole input blocks>,
 86104 <number of partial input blocks>

86105 "%u+%u records out\n", <number of whole output blocks>,
 86106 <number of partial output blocks>

86107 A partial input block is one for which *read()* returned less than the input block size. A partial
 86108 output block is one that was written with fewer bytes than specified by the output block size.

86109 In addition, when there is at least one truncated block, the number of truncated blocks shall be
 86110 written to standard error. In the POSIX locale, the format shall be:

86111 "%u truncated %s\n", <number of truncated blocks>, "record" (if
 86112 <number of truncated blocks> is one) "records" (otherwise)

86113 Diagnostic messages may also be written to standard error.

86114 **OUTPUT FILES**

86115 If the **of=** operand is used, the output shall be the same as described in the STDOUT section.

86116 **EXTENDED DESCRIPTION**

86117 None.

86118 **EXIT STATUS**

86119 The following exit values shall be returned:

86120 0 The input file was copied successfully.

86121 >0 An error occurred.

86122 **CONSEQUENCES OF ERRORS**

86123 If an input error is detected and the **noerror** conversion has not been specified, any partial
86124 output block shall be written to the output file, a diagnostic message shall be written, and the
86125 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall
86126 be written and the copy operation shall be discontinued.

86127 **APPLICATION USAGE**

86128 The input and output block size can be specified to take advantage of raw physical I/O.

86129 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions
86130 specified for the *dd* utility perform conversions for the version specified by the tables.

86131 **EXAMPLES**

86132 The following command:

```
86133 dd if=/dev/rmt0h of=/dev/rmt1h
```

86134 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

86135 The following command:

```
86136 dd ibs=10 skip=1
```

86137 strips the first 10 bytes from standard input.

86138 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the
86139 ASCII file *x*:

```
86140 dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

86141 **RATIONALE**

86142 The **OPTIONS** section is listed as “None” because there are no options recognized by historical
86143 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax
86144 Guidelines, which would have resulted in the classic hyphenated option letters. In this version
86145 of this volume of POSIX.1-2017, *dd* retains its curious JCL-like syntax due to the large number of
86146 applications that depend on the historical implementation.

86147 A suggested implementation technique for **conv=noerror,sync** is to zero (or <space>-fill, if
86148 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input
86149 buffer to the output even after an error. In this manner, any data transferred to the input buffer
86150 before the error was detected is preserved. Another point is that a failed read on a regular file or
86151 a disk generally does not increment the file offset, and *dd* must then seek past the block on which
86152 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic
86153 tape, however, the tape normally has passed the block containing the error when the error is
86154 reported, and thus no seek is necessary.

86155 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely
86156 portable) scripts that assume these values. If they were left unspecified, unusual results could

86157 occur if an implementation chose an odd block size.

86158 Historical implementations of *dd* used *creat()* when processing *of=file*. This makes the **seek=**
86159 operand unusable except on special files. The **conv=notrunc** feature was added because more
86160 recent BSD-based implementations use *open()* (without `O_TRUNC`) instead of *creat()*, but they
86161 fail to delete output file contents after the data copied.

86162 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to
86163 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of
86164 POSIX.1-2017.

86165 Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are
86166 taken from a common print train that does contain them. Other than those characters, the print
86167 train values are not filled in, but appear to provide some of the motivation for the historical
86168 choice of translations reflected here.

86169 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

86170 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ
86171 in such a way that:

- 86172 1. EBCDIC 0112 ('ϕ') and 0152 (broken pipe) do not appear in the table.
- 86173 2. EBCDIC 0137 (' ') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC
86174 0232 (no graphic) is used.
- 86175 3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC
86176 0137 (' ') is used.
- 86177 4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table
86178 and once in place of 0112 ('ϕ') and 0241 ('~').

86179 In net result:

86180 EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.

86181 That displaced EBCDIC 0137 (' ') in cell 0176.

86182 That displaced EBCDIC 0232 (no graphic) in cell 0136.

86183 That replaced EBCDIC 0152 (broken pipe) in cell 0313.

86184 EBCDIC 0255 ('[') replaced EBCDIC 0112 ('ϕ').

86185 This translation, however, reflects historical practice that (ASCII) '~' and ' ' were often
86186 mapped to each other, as were '[' and 'ϕ'; and ']' and (EBCDIC) '~'.

86187 The **cbs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the
86188 **ascii** operand, the input is handled as described for the **unblock** operand except that characters
86189 are converted to ASCII before the trailing <space> characters are deleted. For the **ebcdic** and
86190 **ibm** operands, the input is handled as described for the **block** operand except that the characters
86191 are converted to EBCDIC or IBM EBCDIC after the trailing <space> characters are added.

86192 The **block** and **unblock** keywords are from historical BSD practice.

86193 The consistent use of the word **record** in standard error messages matches most historical
86194 practice. An earlier version of System V used **block**, but this has been updated in more recent
86195 releases.

86196 Early proposals only allowed two numbers separated by *x* to be used in a product when
86197 specifying **bs=**, **cbs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of

- 86198 allowing multiple numbers in the product as provided by Version 7 and all releases of System V
86199 and BSD.
- 86200 A change to the **swab** conversion is required to match historical practice and is the result of IEEE
86201 PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.
- 86202 A change to the handling of SIGINT is required to match historical practice and is the result of
86203 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.
- 86204 **FUTURE DIRECTIONS**
- 86205 None.
- 86206 **SEE ALSO**
- 86207 [Section 1.4](#) (on page 2336), *sed*, *tr*
- 86208 XBD [Chapter 8](#) (on page 173)
- 86209 **CHANGE HISTORY**
- 86210 First released in Issue 2.
- 86211 **Issue 5**
- 86212 The second paragraph of the **cbs=** description is reworded and marked EX.
- 86213 The FUTURE DIRECTIONS section is added.
- 86214 **Issue 6**
- 86215 Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b
86216 draft standard.
- 86217 The normative text is reworded to avoid use of the term “must” for application requirements.
- 86218 IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between **dd of=file** and
86219 **conv=notrunc**.
- 86220 **Issue 7**
- 86221 Austin Group Interpretation 1003.1-2001 #102 is applied.
- 86222 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 86223 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0081 [907] is applied.

86224 NAME

86225 delta ‡make a delta (change) to an SCCS file **DEVELOPMENT**)

86226 SYNOPSIS

86227 xSI delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...

86228 DESCRIPTION

86229 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that
86230 were made to the files retrieved by *get* (called the *g-files*, or generated files).

86231 OPTIONS

86232 The *delta* utility shall conform to XBD Section 12.2 (on page 216), except that the *-y* option has an
86233 optional option-argument. This optional option-argument shall not be presented as a separate
86234 argument.

86235 The following options shall be supported:

86236 *-r SID* Uniquely identify which delta is to be made to the SCCS file. The use of this option
86237 shall be necessary only if two or more outstanding *get* commands for editing (*get*
86238 *-e*) on the same SCCS file were done by the same person (login name). The SID
86239 value specified with the *-r* option can be either the SID specified on the *get*
86240 command line or the SID to be made as reported by the *get* utility; see *get* (on page
86241 2823).86242 *-s* Suppress the report to standard output of the activity associated with each *file*. See
86243 the STDOUT section.86244 *-n* Specify retention of the edited *g-file* (normally removed at completion of delta
86245 processing).86246 *-g list* Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when
86247 the file is accessed at the change level (SID) created by this delta.86248 *-m mrlist* Specify a modification request (MR) number that the application shall supply as
86249 the reason for creating the new delta. This shall be used if the SCCS file has the *v*
86250 flag set; see *admin*.86251 If *-m* is not used and *'-'* is not specified as a file argument, and the standard
86252 input is a terminal, the prompt described in the STDOUT section shall be written
86253 to standard output before the standard input is read; if the standard input is not a
86254 terminal, no prompt shall be issued.86255 MRs in a list shall be separated by <blank> characters or escaped <newline>
86256 characters. An unescaped <newline> shall terminate the MR list. The escape
86257 character is <backslash>.86258 If the *v* flag has a value, it shall be taken to be the name of a program which
86259 validates the correctness of the MR numbers. If a non-zero exit status is returned
86260 from the MR number validation program, the *delta* utility shall terminate. (It is
86261 assumed that the MR numbers were not all valid.)86262 *-y[comment]* Describe the reason for making the delta. The *comment* shall be an arbitrary group
86263 of lines that would meet the definition of a text file. Implementations shall support
86264 *comments* from zero to 512 bytes and may support longer values. A null string
86265 (specified as either *-y*, *-y" "*, or in response to a prompt for a comment) shall be
86266 considered a valid *comment*.86267 If *-y* is not specified and *'-'* is not specified as a file argument, and the standard

86268 input is a terminal, the prompt described in the STDOUT section shall be written
 86269 to standard output before the standard input is read; if the standard input is not a
 86270 terminal, no prompt shall be issued. An unescaped <newline> shall terminate the
 86271 comment text. The escape character is <backslash>.

86272 The `-y` option shall be required if the *file* operand is specified as `'-'`.

86273 **-p** Write (to standard output) the SCCS file differences before and after the delta is
 86274 applied in *diff* format; see *diff*.

86275 OPERANDS

86276 The following operand shall be supported:

86277 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*
 86278 utility shall behave as though each file in the directory were specified as a named
 86279 file, except that non-SCCS files (last component of the pathname does not begin
 86280 with **s**.) and unreadable files shall be silently ignored.

86281 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;
 86282 each line of the standard input shall be taken to be the name of an SCCS file to be
 86283 processed. Non-SCCS files and unreadable files shall be silently ignored.

86284 STDIN

86285 The standard input shall be a text file used only in the following cases:

86286 To read an *mrlist* or a *comment* (see the `-m` and `-y` options).

86287 A *file* operand shall be specified as `'-'`. In this case, the `-y` option must be used to specify
 86288 the comment, and if the SCCS file has the **v** flag set, the `-m` option must also be used to
 86289 specify the MR list.

86290 INPUT FILES

86291 Input files shall be text files whose data is to be included in the SCCS files. If the first character of
 86292 any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file
 86293 contains more than 99 999 lines, the number of lines recorded in the header for this file shall be
 86294 99 999 for this delta.

86295 ENVIRONMENT VARIABLES

86296 The following environment variables shall affect the execution of *delta*:

86297 **LANG** Provide a default value for the internationalization variables that are unset or null.
 86298 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 86299 variables used to determine the values of locale categories.)

86300 **LC_ALL** If set to a non-empty string value, override the values of all the other
 86301 internationalization variables.

86302 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 86303 characters (for example, single-byte as opposed to multi-byte characters in
 86304 arguments and input files).

86305 **LC_MESSAGES**

86306 Determine the locale that should be used to affect the format and contents of
 86307 diagnostic messages written to standard error, and informative messages written
 86308 to standard output.

86309 **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

86310 *TZ* Determine the timezone in which the time and date are written in the SCCS file. If
86311 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

86312 **ASYNCHRONOUS EVENTS**

86313 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit
86314 code. The standard action shall be taken for all other signals; see [Section 1.4](#) (on page 2336).

86315 **STDOUT**

86316 The standard output shall be used only for the following messages in the POSIX locale:

86317 Prompts (see the *-m* and *-y* options) in the following formats:

86318 "MRs? "

86319 "comments? "

86320 The MR prompt, if written, shall always precede the comments prompt.

86321 A report of each file's activities (unless the *-s* option is specified) in the following format:

86322 "%s\n%d inserted\n%d deleted\n%d unchanged\n", *<New SID>*,
86323 *<number of lines inserted>*, *<number of lines deleted>*,
86324 *<number of lines unchanged>*

86325 **STDERR**

86326 The standard error shall be used only for diagnostic messages.

86327 **OUTPUT FILES**

86328 Any SCCS files updated shall be files of an unspecified format.

86329 **EXTENDED DESCRIPTION**

86330 **System Date and Time**

86331 When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new
86332 delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the
86333 behavior is unspecified.

86334 **EXIT STATUS**

86335 The following exit values shall be returned:

86336 0 Successful completion.

86337 >0 An error occurred.

86338 **CONSEQUENCES OF ERRORS**

86339 Default.

86340 **APPLICATION USAGE**

86341 Problems can arise if the system date and time have been modified (for example, put forward
86342 and then back again, or unsynchronized clocks across a network) and can also arise when
86343 different values of the *TZ* environment variable are used.

86344 Problems of a similar nature can also arise for the operation of the *get* utility, which records the
86345 date and time in the file body.

86346 **EXAMPLES**

86347 None.

86348 **RATIONALE**

86349 None.

86350 **FUTURE DIRECTIONS**

86351 None.

86352 **SEE ALSO**86353 [Section 1.4](#) (on page 2336), *admin*, *diff*, *get*, *prs*, *rmdel*86354 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)86355 **CHANGE HISTORY**

86356 First released in Issue 2.

86357 **Issue 5**

86358 The output format description in the STDOUT section is corrected.

86359 **Issue 6**

86360 The APPLICATION USAGE section is added.

86361 The normative text is reworded to avoid use of the term “must” for application requirements.

86362 The Open Group Base Resolution bwg2001-007 is applied as follows:

86363 The use of '-' as a file argument is clarified.

86364 The use of STDIN is added.

86365 The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement
86366 that implementations re-signal themselves when catching a normally fatal signal.86367 New text is added to the INPUT FILES section warning that the maximum lines recorded
86368 in the file is 99 999.86369 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections
86370 regarding how the system date and time may be taken into account, and the TZ environment
86371 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base
86372 Resolution bwg2001-007.86373 **Issue 7**

86374 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

86375 **NAME**

86376 df — report free disk space

86377 **SYNOPSIS**86378 XSI df [-k] [-P|-t] [*file...*]86379 **DESCRIPTION**

86380 XSI The *df* utility shall write the amount of available space and file slots for file systems on which
 86381 the invoking user has appropriate read access. File systems shall be specified by the *file*
 86382 operands; when none are specified, information shall be written for all file systems. The format
 86383 of the default output from *df* is unspecified, but all space figures are reported in 512-byte units,
 86384 unless the **-k** option is specified. This output shall contain at least the file system names, amount
 86385 XSI of available space on each of these file systems, and, if no options other than **-t** are specified, the
 86386 number of free file slots, or *inodes*, available; when **-t** is specified, the output shall contain the
 86387 total allocated space as well.

86388 **OPTIONS**86389 The *df* utility shall conform to XBD Section 12.2 (on page 216).

86390 The following options shall be supported:

86391 **-k** Use 1024-byte units, instead of the default 512-byte units, when writing space
 86392 figures.

86393 **-P** Produce output in the format described in the STDOUT section.

86394 XSI **-t** Include total allocated-space figures in the output.

86395 **OPERANDS**

86396 The following operand shall be supported:

86397 *file* A pathname of a file within the hierarchy of the desired file system. If a file other
 86398 XSI than a FIFO, a regular file, a directory, or a special file representing the device
 86399 containing the file system (for example, **/dev/dsk/0s1**) is specified, the results are
 86400 unspecified. If the *file* operand names a file other than a special file containing a file
 86401 system, *df* shall write the amount of free space in the file system containing the
 86402 XSI specified *file* operand. Otherwise, *df* shall write the amount of free space in that
 86403 file system.

86404 **STDIN**

86405 Not used.

86406 **INPUT FILES**

86407 None.

86408 **ENVIRONMENT VARIABLES**86409 The following environment variables shall affect the execution of *df*:

86410 **LANG** Provide a default value for the internationalization variables that are unset or null.
 86411 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 86412 variables used to determine the values of locale categories.)

86413 **LC_ALL** If set to a non-empty string value, override the values of all the other
 86414 internationalization variables.

86415 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 86416 characters (for example, single-byte as opposed to multi-byte characters in
 86417 arguments).

86418 *LC_MESSAGES*
 86419 Determine the locale that should be used to affect the format and contents of
 86420 diagnostic messages written to standard error and informative messages written to
 86421 standard output.

86422 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

86423 **ASYNCHRONOUS EVENTS**
 86424 Default.

86425 **STDOUT**
 86426 When both the **-k** and **-P** options are specified, the following header line shall be written (in the
 86427 POSIX locale):
 86428 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"
 86429 When the **-P** option is specified without the **-k** option, the following header line shall be written
 86430 (in the POSIX locale):
 86431 "Filesystem 512-blocks Used Available Capacity Mounted on\n"
 86432 The implementation may adjust the spacing of the header line and the individual data lines so
 86433 that the information is presented in orderly columns.
 86434 The remaining output with **-P** shall consist of one line of information for each specified file
 86435 system. These lines shall be formatted as follows:
 86436 "%s %d %d %d %d%% %s\n", *<file system name>*, *<total space>*,
 86437 *<space used>*, *<space free>*, *<percentage used>*,
 86438 *<file system root>*
 86439 In the following list, all quantities expressed in 512-byte units (1 024-byte when **-k** is specified)
 86440 shall be rounded up to the next higher unit. The fields are:
 86441 *<file system name>*
 86442 The name of the file system, in an implementation-defined format.
 86443 *<total space>* The total size of the file system in 512-byte units. The exact meaning of this figure
 86444 is implementation-defined, but should include *<space used>*, *<space free>*, plus any
 86445 space reserved by the system not normally available to a user.
 86446 *<space used>* The total amount of space allocated to existing files in the file system, in 512-byte
 86447 units.
 86448 *<space free>* The total amount of space available within the file system for the creation of new
 86449 files by unprivileged users, in 512-byte units. When this figure is less than or equal
 86450 to zero, it shall not be possible to create any new files on the file system without
 86451 first deleting others, unless the process has appropriate privileges. The figure
 86452 written may be less than zero.
 86453 *<percentage used>*
 86454 The percentage of the normally available space that is currently allocated to all files
 86455 on the file system. This shall be calculated using the fraction:
 86456
$$\frac{\textit{<space used>}}{\textit{<space used> + <space free>}}$$

 86457 expressed as a percentage. This percentage may be greater than 100 if *<space free>*
 86458 is less than zero. The percentage value shall be expressed as a positive integer, with
 86459 any fractional result causing it to be rounded to the next highest integer.

86460 <*file system root*>
 86461 The directory below which the file system hierarchy appears.

86462 XSI The output format is unspecified when `-t` is used.

86463 **STDERR**
 86464 The standard error shall be used only for diagnostic messages.

86465 **OUTPUT FILES**
 86466 None.

86467 **EXTENDED DESCRIPTION**
 86468 None.

86469 **EXIT STATUS**
 86470 The following exit values shall be returned:
 86471 0 Successful completion.
 86472 >0 An error occurred.

86473 **CONSEQUENCES OF ERRORS**
 86474 Default.

86475 **APPLICATION USAGE**
 86476 On most systems, the “name of the file system, in an implementation-defined format” is the
 86477 special file on which the file system is mounted.
 86478 On large file systems, the calculation specified for percentage used can create huge rounding
 86479 errors.

86480 **EXAMPLES**
 86481 1. The following example writes portable information about the `/usr` file system:
 86482 `df -P /usr`
 86483 2. Assuming that `/usr/src` is part of the `/usr` file system, the following produces the same
 86484 output as the previous example:
 86485 `df -P /usr/src`

86486 **RATIONALE**
 86487 The behavior of `df` with the `-P` option is the default action of the 4.2 BSD `df` utility. The uppercase
 86488 `-P` was selected to avoid collision with a known industry extension using `-p`.
 86489 Historical `df` implementations vary considerably in their default output. It was therefore
 86490 necessary to describe the default output in a loose manner to accommodate all known historical
 86491 implementations and to add a portable option (`-P`) to provide information in a portable format.
 86492 The use of 512-byte units is historical practice and maintains compatibility with `ls` and other
 86493 utilities in this volume of POSIX.1-2017. This does not mandate that the file system itself be
 86494 based on 512-byte blocks. The `-k` option was added as a compromise measure. It was agreed by
 86495 the standard developers that 512 bytes was the best default unit because of its complete
 86496 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and
 86497 that a `-k` option to switch to 1024-byte units was a good compromise. Users who prefer the
 86498 more logical 1024-byte quantity can easily alias `df` to `df -k` without breaking many historical
 86499 scripts relying on the 512-byte units.
 86500 It was suggested that `df` and the various related utilities be modified to access a `BLOCKSIZE`
 86501 environment variable to achieve consistency and user acceptance. Since this is not historical

86502 practice on any system, it is left as a possible area for system extensions and will be re-evaluated
86503 in a future version if it is widely implemented.

86504 **FUTURE DIRECTIONS**

86505 None.

86506 **SEE ALSO**

86507 *find*

86508 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

86509 **CHANGE HISTORY**

86510 First released in Issue 2.

86511 **Issue 6**

86512 This utility is marked as part of the User Portability Utilities option.

86513 **Issue 7**

86514 Austin Group Interpretation 1003.1-2001 #099 is applied.

86515 The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is
86516 now an option for interactive utilities.

86517 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

86518 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0082 [156] is applied.

86519 **NAME**

86520 diff compare two files

86521 **SYNOPSIS**

86522 diff [-c|-e|-f|-u|-C n|-U n] [-br] file1 file2

86523 **DESCRIPTION**

86524 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of
 86525 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be
 86526 produced if the files are identical.

86527 **OPTIONS**86528 The *diff* utility shall conform to XBD [Section 12.2](#) (on page 216).

86529 The following options shall be supported:

86530 **-b** Cause any amount of white space at the end of a line to be treated as a single
 86531 <newline> (that is, the white-space characters preceding the <newline> are
 86532 ignored) and other strings of white-space characters, not including <newline>
 86533 characters, to compare equal.

86534 **-c** Produce output in a form that provides three lines of copied context.

86535 **-C n** Produce output in a form that provides *n* lines of copied context (where *n* shall be
 86536 interpreted as a positive decimal integer).

86537 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be used
 86538 to convert *file1* into *file2*.

86539 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to
 86540 be suitable as input for the *ed* utility, and in the opposite order.

86541 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*
 86542 are both directories.

86543 The *diff* utility shall detect infinite loops; that is, entering a previously visited
 86544 directory that is an ancestor of the last file encountered. When it detects an infinite
 86545 loop, *diff* shall write a diagnostic message to standard error and shall either recover
 86546 its position in the hierarchy or terminate.

86547 **-u** Produce output in a form that provides three lines of unified context.

86548 **-U n** Produce output in a form that provides *n* lines of unified context (where *n* shall be
 86549 interpreted as a non-negative decimal integer).

86550 **OPERANDS**

86551 The following operands shall be supported:

86552 *file1, file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the
 86553 standard input shall be used in its place.

86554 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special
 86555 files, or FIFO special files to any files and shall not compare regular files to directories. Further
 86556 details are as specified in [Diff Directory Comparison Format](#) (on page 2659). The behavior of *diff*
 86557 on other file types is implementation-defined when found in directories.

86558 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file
 86559 contained in the directory file with a filename that is the same as the last component of the non-
 86560 directory file.

86561 **STDIN**

86562 The standard input shall be used only if one of the *file1* or *file2* operands references standard
86563 input. See the INPUT FILES section.

86564 **INPUT FILES**

86565 The input files may be of any type.

86566 **ENVIRONMENT VARIABLES**

86567 The following environment variables shall affect the execution of *diff*:

86568 *LANG* Provide a default value for the internationalization variables that are unset or null.
86569 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
86570 variables used to determine the values of locale categories.)

86571 *LC_ALL* If set to a non-empty string value, override the values of all the other
86572 internationalization variables.

86573 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
86574 characters (for example, single-byte as opposed to multi-byte characters in
86575 arguments and input files).

86576 *LC_MESSAGES*

86577 Determine the locale that should be used to affect the format and contents of
86578 diagnostic messages written to standard error and informative messages written to
86579 standard output.

86580 *LC_TIME* Determine the locale for affecting the format of file timestamps written with the *-C*
86581 and *-c* options.

86582 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

86583 *TZ* Determine the timezone used for calculating file timestamps written with a context
86584 format. If *TZ* is unset or null, an unspecified default timezone shall be used.

86585 **ASYNCHRONOUS EVENTS**

86586 Default.

86587 **STDOUT**86588 **Diff Directory Comparison Format**

86589 If both *file1* and *file2* are directories, the following output formats shall be used.

86590 In the POSIX locale, each file that is present in only one directory shall be reported using the
86591 following format:

86592 "Only in %s: %s\n", <directory pathname>, <filename>

86593 In the POSIX locale, subdirectories that are common to the two directories may be reported with
86594 the following format:

86595 "Common subdirectories: %s and %s\n", <directory1 pathname>,
86596 <directory2 pathname>

86597 For each file common to the two directories, if the two files are not to be compared: if the two
86598 files have the same device ID and file serial number, or are both block special files that refer to
86599 the same device, or are both character special files that refer to the same device, in the POSIX
86600 locale the output format is unspecified. Otherwise, in the POSIX locale an unspecified format
86601 shall be used that contains the pathnames of the two files.

86602 For each file common to the two directories, if the files are compared and are identical, no

86603 output shall be written. If the two files differ, the following format is written:

86604 "diff %s %s %s\n", <diff_options>, <filename1>, <filename2>

86605 where <diff_options> are the options as specified on the command line.

86606 All directory pathnames listed in this section shall be relative to the original command line
86607 arguments. All other names of files listed in this section shall be filenames (pathname
86608 components).

86609 Diff Binary Output Format

86610 In the POSIX locale, if one or both of the files being compared are not text files, it is
86611 implementation-defined whether *diff* uses the binary file output format or the other formats as
86612 specified below. The binary file output format shall contain the pathnames of two files being
86613 compared and the string "differ".

86614 If both files being compared are text files, depending on the options specified, one of the
86615 following formats shall be used to write the differences.

86616 Diff Default Output Format

86617 The default (without *-e*, *-f*, *-c*, *-C*, *-u*, or *-U* options) *diff* utility output shall contain lines of
86618 these forms:

86619 "%da%d\n", <num1>, <num2>

86620 "%da%d,%d\n", <num1>, <num2>, <num3>

86621 "%dd%d\n", <num1>, <num2>

86622 "%d,%dd%d\n", <num1>, <num2>, <num3>

86623 "%dc%d\n", <num1>, <num2>

86624 "%d,%dc%d\n", <num1>, <num2>, <num3>

86625 "%dc%d,%d\n", <num1>, <num2>, <num3>

86626 "%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>

86627 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the
86628 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*
86629 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in
86630 *ed*, identical pairs (where *num1*= *num2*) are abbreviated as a single number.

86631 Following each of these lines, *diff* shall write to standard output all lines affected in the first file
86632 using the format:

86633 "<Δ%s", <line>

86634 and all lines affected in the second file using the format:

86635 ">Δ%s", <line>

86636 If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are
86637 separated with a line consisting of three <hyphen-minus> characters:

86638 "---\n"

86639 **Diff -e Output Format**

86640 With the `-e` option, a script shall be produced that shall, when provided as input to `ed`, along
 86641 with an appended `w` (write) command, convert *file1* into *file2*. Only the `a` (append), `c` (change), `d`
 86642 (delete), `i` (insert), and `s` (substitute) commands of `ed` shall be used in this script. Text lines,
 86643 except those consisting of the single character `<period>` (`'.'`), shall be output as they appear in
 86644 the file.

86645 **Diff -f Output Format**

86646 With the `-f` option, an alternative format of script shall be produced. It is similar to that
 86647 produced by `-e`, with the following differences:

- 86648 1. It is expressed in reverse sequence; the output of `-e` orders changes from the end of the
 86649 file to the beginning; the `-f` from beginning to end.
- 86650 2. The command form `<lines> <command-letter>` used by `-e` is reversed. For example,
 86651 `10c` with `-e` would be `c10` with `-f`.
- 86652 3. The form used for ranges of line numbers is `<space>`-separated, rather than
 86653 `<comma>`-separated.

86654 **Diff -c or -C Output Format**

86655 With the `-c` or `-C` option, the output format shall consist of affected lines along with
 86656 surrounding lines of context. The affected lines shall show which ones need to be deleted or
 86657 changed in *file1*, and those added from *file2*. With the `-c` option, three lines of context, if
 86658 available, shall be written before and after the affected lines. With the `-C` option, the user can
 86659 specify how many lines of context are written. The exact format follows.

86660 The name and last modification time of each file shall be output in the following format:

```
86661 "*** %s %s\n", file1, <file1 timestamp>
86662 "--- %s %s\n", file2, <file2 timestamp>
```

86663 Each `<file>` field shall be the pathname of the corresponding file being compared. The pathname
 86664 written for standard input is unspecified.

86665 In the POSIX locale, each `<timestamp>` field shall be equivalent to the output from the following
 86666 command:

```
86667 date "+%a %b %e %T %Y"
```

86668 without the trailing `<newline>`, executed at the time of last modification of the corresponding
 86669 file (or the current time, if the file is standard input).

86670 Then, the following output formats shall be applied for every set of changes.

86671 First, a line shall be written in the following format:

```
86672 "*****\n"
```

86673 Next, the range of lines in *file1* shall be written in the following format if the range contains two
 86674 or more lines:

```
86675 "*** %d,%d ****\n", <beginning line number>, <ending line number>
```

86676 and the following format otherwise:

```
86677 "*** %d ****\n", <ending line number>
```

86678 The ending line number of an empty range shall be the number of the preceding line, or 0 if the

86679 range is at the start of the file.

86680 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected
86681 lines shall be written in the following format:

86682 " Δ %s", <unaffected_line>

86683 Deleted lines shall be written as:

86684 " $-\Delta$ %s", <deleted_line>

86685 Changed lines shall be written as:

86686 " $!\Delta$ %s", <changed_line>

86687 Next, the range of lines in *file2* shall be written in the following format if the range contains two
86688 or more lines:

86689 "--- %d,%d ----\n", <beginning line number>, <ending line number>

86690 and the following format otherwise:

86691 "--- %d ----\n", <ending line number>

86692 Then, lines of context and changed lines shall be written as described in the previous formats.
86693 Lines added from *file2* shall be written in the following format:

86694 "+ Δ %s", <added_line>

86695 **Diff $-u$ or $-U$ Output Format**

86696 The $-u$ or $-U$ options behave like the $-c$ or $-C$ options, except that the context lines are not
86697 repeated; instead, the context, deleted, and added lines are shown together, interleaved. The
86698 exact format follows.

86699 The name and last modification time of each file shall be output in the following format:

86700 "--- Δ %s\t%s%s Δ %s\n", file1, <file1 timestamp>, <file1 frac>, <file1 zone>
86701 "+++ Δ %s\t%s%s Δ %s\n", file2, <file2 timestamp>, <file2 frac>, <file2 zone>

86702 Each <file> field shall be the pathname of the corresponding file being compared, or the single
86703 character '-' if standard input is being compared. However, if the pathname contains a <tab>
86704 or a <newline>, or if it does not consist entirely of characters taken from the portable character
86705 set, the behavior is implementation-defined.

86706 Each <timestamp> field shall be equivalent to the output from the following command:

86707 date '+%Y-%m-%d Δ %H:%M:%S'

86708 without the trailing <newline>, executed at the time of last modification of the corresponding
86709 file (or the current time, if the file is standard input).

86710 Each <frac> field shall be either empty, or a decimal point followed by at least one decimal digit,
86711 indicating the fractional-seconds part (if any) of the file timestamp. The number of fractional
86712 digits shall be at least the number needed to represent the file's timestamp without loss of
86713 information.

86714 Each <zone> field shall be of the form "shhmm", where "shh" is a signed two-digit decimal
86715 number in the range -24 through +25, and "mm" is an unsigned two-digit decimal number in the
86716 range 00 through 59. It represents the timezone of the timestamp as the number of hours (hh)
86717 and minutes (mm) east (+) or west (-) of UTC for the timestamp. If the hours and minutes are
86718 both zero, the sign shall be '+'. However, if the timezone is not an integral number of minutes

86719 away from UTC, the `<zone>` field is implementation-defined.

86720 Then, the following output formats shall be applied for every set of changes.

86721 First, the range of lines in each file shall be written in the following format:

86722 "@@Δ-`%s`Δ+`%s`Δ@@", `<file1 range>`, `<file2 range>`

86723 Each `<range>` field shall be of the form:

86724 "`%ld`", `<beginning line number>`

86725 or:

86726 "`%ld,1`", `<beginning line number>`

86727 if the range contains exactly one line, and:

86728 "`%ld,%ld`", `<beginning line number>`, `<number of lines>`

86729 otherwise. If a range is empty, its beginning line number shall be the number of the line just
86730 before the range, or 0 if the empty range starts the file.

86731 Next, the affected lines along with lines of context shall be written. Each non-empty unaffected
86732 line shall be written in the following format:

86733 "Δ`%s`", `<unaffected_line>`

86734 where the contents of the unaffected line shall be taken from *file1*. It is implementation-defined
86735 whether an empty unaffected line is written as an empty line or a line containing a single
86736 `<space>` character. This line also represents the same line of *file2*, even though *file2*'s line may
86737 contain different contents due to the `-b`. Deleted lines shall be written as:

86738 "-`%s`", `<deleted_line>`

86739 Added lines shall be written as:

86740 "+`%s`", `<added_line>`

86741 The order of lines written shall be the same as that of the corresponding file. A deleted line shall
86742 never be written immediately after an added line.

86743 If `-U n` is specified, the output shall contain no more than $2n$ consecutive unaffected lines; and if
86744 the output contains an affected line and this line is adjacent to up to n consecutive unaffected
86745 lines in the corresponding file, the output shall contain these unaffected lines. `-u` shall act like
86746 `-U3`.

86747 **STDERR**

86748 The standard error shall be used only for diagnostic messages.

86749 **OUTPUT FILES**

86750 None.

86751 **EXTENDED DESCRIPTION**

86752 None.

86753 **EXIT STATUS**

86754 The following exit values shall be returned:

86755 0 No differences were found.

86756 1 Differences were found.

86757 >1 An error occurred.

86758 CONSEQUENCES OF ERRORS

86759 Default.

86760 APPLICATION USAGE

86761 If lines at the end of a file are changed and other lines are added, *diff* output may show this as a
 86762 delete and add, as a change, or as a change and add; *diff* is not expected to know which
 86763 happened and users should not care about the difference in output as long as it clearly shows
 86764 the differences between the files.

86765 EXAMPLES

86766 If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory
 86767 named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**,
 86768 the command:

```
86769 diff -r dir1 dir2
```

86770 could produce output similar to:

```
86771 Common subdirectories: dir1/x and dir2/x
86772 Only in dir2/x: y
86773 diff -r dir1/x/date.out dir2/x/date.out
86774 1c1
86775 < Mon Jul  2 13:12:16 PDT 1990
86776 ---
86777 > Tue Jun 19 21:41:39 PDT 1990
```

86778 RATIONALE

86779 The **-h** option was omitted because it was insufficiently specified and does not add to
 86780 applications portability.

86781 Historical implementations employ algorithms that do not always produce a minimum list of
 86782 differences; the current language about making every effort is the best this volume of
 86783 POSIX.1-2017 can do, as there is no metric that could be employed to judge the quality of
 86784 implementations against any and all file contents. The statement “This list should be minimal”
 86785 clearly implies that implementations are not expected to provide the following output when
 86786 comparing two 100-line files that differ in only one character on a single line:

```
86787 1,100c1,100
86788 all 100 lines from file1 preceded with "< "
86789 ---
86790 all 100 lines from file2 preceded with "> "
```

86791 The “Only in” messages required when the **-r** option is specified are not used by most historical
 86792 implementations if the **-e** option is also specified. It is required here because it provides useful
 86793 information that must be provided to update a target directory hierarchy to match a source
 86794 hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when
 86795 the **-r** option is specified. They are allowed here but are not required because they are reporting
 86796 on something that is the same, not reporting a difference, and are not needed to update a target
 86797 hierarchy.

86798 The **-c** option, which writes output in a format using lines of context, has been included. The
 86799 format is useful for a variety of reasons, among them being much improved readability and the
 86800 ability to understand difference changes when the target file has line numbers that differ from
 86801 another similar, but slightly different, copy. The *patch* utility is most valuable when working
 86802 with difference listings using a context format. The BSD version of **-c** takes an optional

86803 argument specifying the amount of context. Rather than overloading `-c` and breaking the Utility
86804 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for
86805 specifying a context *diff* with a specified amount of context (`-C`). Also, the format for context
86806 *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines
86807 from each other to be merged together. The output format contains an additional four `<asterisk>`
86808 characters after the range of affected lines in the first filename. This was to provide a flag for old
86809 programs (like old versions of *patch*) that only understand the old context format. The version of
86810 context described here does not require that multiple changes within context lines be merged,
86811 but it does not prohibit it either. The extension is upwards-compatible, so any vendors that wish
86812 to retain the old version of *diff* can do so by adding the extra four `<asterisk>` characters (that is,
86813 utilities that currently use *diff* and understand the new merged format will also understand the
86814 old unmerged format, but not *vice versa*).

86815 The `-u` and `-U` options of GNU *diff* have been included. Their output format, designed by
86816 Wayne Davison, takes up less space than `-c` and `-C` format, and in many cases is easier to read.
86817 The format's timestamps do not vary by locale, so `LC_TIME` does not affect it. The format's line
86818 numbers are rendered with the `%ld` format, not `%d`, because the file format notation rules would
86819 allow extra `<blank>` characters to appear around the numbers.

86820 The substitute command was added as an additional format for the `-e` option. This was added
86821 to provide implementations with a way to fix the classic ```dot alone on a line``` bug present in
86822 many versions of *diff*. Since many implementations have fixed this bug, the standard developers
86823 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing
86824 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then
86825 terminate the append command with a period, and then use the substitute command to convert
86826 the two periods into one period.

86827 The BSD-derived `-r` option was added to provide a mechanism for using *diff* to compare two file
86828 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not
86829 easily reproducible with the *find* utility.

86830 The requirement that *diff* not compare files in some circumstances, even though they have the
86831 same name, is based on the actual output of historical implementations. The specified behavior
86832 precludes the problems arising from running into FIFOs and other files that would cause *diff*
86833 to hang waiting for input with no indication to the user that *diff* was hung. An earlier version of
86834 this standard specified the output format more precisely, but in practice this requirement was
86835 widely ignored and the benefit of standardization seemed small, so it is now unspecified. In
86836 most common usage, *diff -r* should indicate differences in the file hierarchies, not the difference
86837 of contents of devices pointed to by the hierarchies.

86838 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of
86839 POSIX.1-2017 supports named pipes, the standard developers decided that such a restriction
86840 was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

86841 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in
86842 that it prints out all of the differences at the current level, including the statements about all
86843 common subdirectories before recursing into those subdirectories.

86844 The message:

```
86845 "diff %s %s %s\n", <diff_options>, <filename1>, <filename2>
```

86846 does not vary by locale because it is the representation of a command, not an English sentence.

86847 **FUTURE DIRECTIONS**

86848 None.

86849 **SEE ALSO**86850 *cmp, comm, ed, find*86851 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)86852 **CHANGE HISTORY**

86853 First released in Issue 2.

86854 **Issue 5**

86855 The FUTURE DIRECTIONS section is added.

86856 **Issue 6**86857 The following new requirements on POSIX implementations derive from alignment with the
86858 Single UNIX Specification:86859 The `-f` option is added.86860 The output format for `-c` or `-C` format is changed to align with changes to the IEEE P1003.2b
86861 draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

86862 The normative text is reworded to avoid use of the term “must” for application requirements.

86863 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT
86864 section. This changes the specification of *diff -c* so that it agrees with existing practice when
86865 contexts contain zero lines or one line.86866 **Issue 7**

86867 Austin Group Interpretations 1003.1-2001 #115 and #114 are applied.

86868 Austin Group Interpretation 1003.1-2001 #192 is applied, clarifying the behavior if both files are
86869 non-text files.

86870 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

86871 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.86872 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0082 [584], XCU/TC2-2008/0083
86873 [950], XCU/TC2-2008/0084 [969], and XCU/TC2-2008/0085 [929] are applied.

86874 **NAME**

86875 dirname — return the directory portion of a pathname

86876 **SYNOPSIS**86877 dirname *string*86878 **DESCRIPTION**

86879 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.271](#) (on page 76).
 86880 The string *string* shall be converted to the name of the directory containing the filename
 86881 corresponding to the last pathname component in *string*, performing actions equivalent to the
 86882 following steps in order:

- 86883 1. If *string* is *//*, skip steps 2 to 5.
- 86884 2. If *string* consists entirely of <slash> characters, *string* shall be set to a single <slash>
 86885 character. In this case, skip steps 3 to 8.
- 86886 3. If there are any trailing <slash> characters in *string*, they shall be removed.
- 86887 4. If there are no <slash> characters remaining in *string*, *string* shall be set to a single
 86888 <period> character. In this case, skip steps 5 to 8.
- 86889 5. If there are any trailing non-<slash> characters in *string*, they shall be removed.
- 86890 6. If the remaining *string* is *//*, it is implementation-defined whether steps 7 and 8 are
 86891 skipped or processed.
- 86892 7. If there are any trailing <slash> characters in *string*, they shall be removed.
- 86893 8. If the remaining *string* is empty, *string* shall be set to a single <slash> character.

86894 The resulting string shall be written to standard output.

86895 **OPTIONS**

86896 None.

86897 **OPERANDS**

86898 The following operand shall be supported:

86899 *string* A string.86900 **STDIN**

86901 Not used.

86902 **INPUT FILES**

86903 None.

86904 **ENVIRONMENT VARIABLES**86905 The following environment variables shall affect the execution of *dirname*:

86906 *LANG* Provide a default value for the internationalization variables that are unset or null.
 86907 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 86908 variables used to determine the values of locale categories.)

86909 *LC_ALL* If set to a non-empty string value, override the values of all the other
 86910 internationalization variables.

86911 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 86912 characters (for example, single-byte as opposed to multi-byte characters in
 86913 arguments).

86914 *LC_MESSAGES*

86915 Determine the locale that should be used to affect the format and contents of

86916 diagnostic messages written to standard error.

86917 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

86918 **ASYNCHRONOUS EVENTS**

86919 Default.

86920 **STDOUT**

86921 The *dirname* utility shall write a line to the standard output in the following format:

86922 "%s\n", <resulting string>

86923 **STDERR**

86924 The standard error shall be used only for diagnostic messages.

86925 **OUTPUT FILES**

86926 None.

86927 **EXTENDED DESCRIPTION**

86928 None.

86929 **EXIT STATUS**

86930 The following exit values shall be returned:

86931 0 Successful completion.

86932 >0 An error occurred.

86933 **CONSEQUENCES OF ERRORS**

86934 Default.

86935 **APPLICATION USAGE**

86936 The definition of *pathname* specifies implementation-defined behavior for pathnames starting

86937 with two <slash> characters. Therefore, applications shall not arbitrarily add <slash> characters

86938 to the beginning of a pathname unless they can ensure that there are more or less than two or are

86939 prepared to deal with the implementation-defined consequences.

86940 **EXAMPLES**

86941 The EXAMPLES section of the *basename()* function (see XSH *basename()*) includes a table

86942 showing examples of the results of processing several sample pathnames by the *basename()* and

86943 *dirname()* functions and by the *basename* and *dirname* utilities.

86944 See also the examples for the *basename* utility.

86945 **RATIONALE**

86946 The behaviors of *basename* and *dirname* in this volume of POSIX.1-2017 have been coordinated so

86947 that when *string* is a valid pathname:

86948 `$(basename -- "string")`

86949 would be a valid filename for the file in the directory:

86950 `$(dirname -- "string")`

86951 This would not work for the versions of these utilities in early proposals due to the way

86952 processing of trailing <slash> characters was specified. Consideration was given to leaving

86953 processing unspecified if there were trailing <slash> characters, but this cannot be done; XBD

86954 [Section 3.271](#) (on page 76) allows trailing <slash> characters. The *basename* and *dirname* utilities

86955 have to specify consistent handling for all valid pathnames.

86956 **FUTURE DIRECTIONS**

86957 None.

86958 **SEE ALSO**86959 [Section 2.5](#) (on page 2349), *basename*86960 [XBD Section 3.271](#) (on page 76), [Chapter 8](#) (on page 173)86961 XSH *basename()*, *dirname()*86962 **CHANGE HISTORY**

86963 First released in Issue 2.

86964 **Issue 7**86965 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0083 [192,430], XCU/TC1-2008/0084
86966 [192], and XCU/TC1-2008/0085 [192] are applied.86967 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0086 [612], XCU/TC2-2008/0087
86968 [620], and XCU/TC2-2008/0088 [612] are applied.

86969 **NAME**

86970 du ‡estimate file space usage

86971 **SYNOPSIS**86972 du [-a|-s] [-kx] [-H|-L] [*file...*]86973 **DESCRIPTION**

86974 By default, the *du* utility shall write to standard output the size of the file space allocated to, and
 86975 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the
 86976 specified files. By default, when a symbolic link is encountered on the command line or in the
 86977 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the
 86978 link), and shall not follow the link to another portion of the file hierarchy. The size of the file
 86979 space allocated to a file of type directory shall be defined as the sum total of space allocated to
 86980 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

86981 When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the
 86982 final exit status is affected. A file that occurs multiple times under one file operand and that has
 86983 a link count greater than 1 shall be counted and written for only one entry. It is implementation-
 86984 defined whether a file that has a link count no greater than 1 is counted and written just once, or
 86985 is counted and written for each occurrence. It is implementation-defined whether a file that
 86986 occurs under one file operand is counted for other file operands. The directory entry that is
 86987 selected in the report is unspecified. By default, file sizes shall be written in 512-byte units,
 86988 rounded up to the next 512-byte unit.

86989 **OPTIONS**86990 The *du* utility shall conform to XBD [Section 12.2](#) (on page 216).

86991 The following options shall be supported:

86992 **-a** In addition to the default output, report the size of each file not of type directory in
 86993 the file hierarchy rooted in the specified file. The **-a** option shall not affect whether
 86994 non-directories given as *file* operands are listed.

86995 **-H** If a symbolic link is specified on the command line, *du* shall count the size of the
 86996 file or file hierarchy referenced by the link.

86997 **-k** Write the files sizes in units of 1024 bytes, rather than the default 512-byte units.

86998 **-L** If a symbolic link is specified on the command line or encountered during the
 86999 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy
 87000 referenced by the link.

87001 **-s** Instead of the default output, report only the total sum for each of the specified
 87002 files.

87003 **-x** When evaluating file sizes, evaluate only those files that have the same device as
 87004 the file specified by the *file* operand.

87005 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
 87006 an error. The last option specified shall determine the behavior of the utility.

87007 **OPERANDS**

87008 The following operand shall be supported:

87009 *file* The pathname of a file whose size is to be written. If no *file* is specified, the current
 87010 directory shall be used.

87011 **STDIN**

87012 Not used.

87013 **INPUT FILES**

87014 None.

87015 **ENVIRONMENT VARIABLES**87016 The following environment variables shall affect the execution of *du*:

87017 *LANG* Provide a default value for the internationalization variables that are unset or null.
 87018 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 87019 variables used to determine the values of locale categories.)

87020 *LC_ALL* If set to a non-empty string value, override the values of all the other
 87021 internationalization variables.

87022 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 87023 characters (for example, single-byte as opposed to multi-byte characters in
 87024 arguments).

87025 *LC_MESSAGES*

87026 Determine the locale that should be used to affect the format and contents of
 87027 diagnostic messages written to standard error.

87028 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

87029 **ASYNCHRONOUS EVENTS**

87030 Default.

87031 **STDOUT**

87032 The output from *du* shall consist of the amount of space allocated to a file and the name of the
 87033 file, in the following format:

87034 "%d %s\n", *<size>*, *<pathname>*87035 **STDERR**

87036 The standard error shall be used only for diagnostic messages.

87037 **OUTPUT FILES**

87038 None.

87039 **EXTENDED DESCRIPTION**

87040 None.

87041 **EXIT STATUS**

87042 The following exit values shall be returned:

87043 0 Successful completion.

87044 >0 An error occurred.

87045 **CONSEQUENCES OF ERRORS**

87046 Default.

87047 **APPLICATION USAGE**

87048 None.

87049 **EXAMPLES**

87050 None.

87051 **RATIONALE**

87052 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
87053 utilities in this volume of POSIX.1-2017. This does not mandate that the file system itself be
87054 based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed by
87055 the standard developers that 512 bytes was the best default unit because of its complete
87056 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and
87057 that a **-k** option to switch to 1024-byte units was a good compromise. Users who prefer the
87058 1024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts
87059 relying on the 512-byte units.

87060 The **-b** option was added to an early proposal to provide a resolution to the situation where
87061 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-
87062 defined concept. (In common usage, the block size is 512 bytes for System V and 1024 bytes for
87063 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as
87064 512-byte units.

87065 Historical file systems provided no way to obtain exact figures for the space allocation given to
87066 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*
87067 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is
87068 space used by the file system in the storage of the file, but that need not be counted in the space
87069 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position
87070 beyond the end of the file and data has subsequently been written at that point. A file system
87071 need not allocate all the intervening zero-filled blocks to such a file. It is up to the
87072 implementation to define exactly how accurate its methods are.

87073 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell
87074 and Utilities description is implied by the language in the SVID where **-s** is described as causing
87075 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly
87076 Conforming POSIX Shell and Utilities Application cannot use that combination.

87077 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not
87078 listing non-directories explicitly given as operands, unless the **-a** option is specified, was
87079 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default
87080 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no
87081 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and
87082 Utilities default behavior shall be to produce such messages. These messages can be turned off
87083 with shell redirection to achieve the System V behavior.

87084 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of
87085 POSIX.1-2017 because there was no other historical method of limiting the *du* search to a single
87086 file hierarchy. This limitation of the search is necessary to make it possible to obtain file space
87087 usage information about a file system on which other file systems are mounted, without having
87088 to resort to a lengthy *find* and *awk* script.

87089 **FUTURE DIRECTIONS**

87090 A future version of this standard may require that a file that occurs multiple times shall be
87091 counted and written for only one entry, even if the occurrences are under different file operands.

87092 **SEE ALSO**87093 [ls](#)87094 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)87095 [XSH *fstatat*\(\)](#)87096 **CHANGE HISTORY**

87097 First released in Issue 2.

87098 **Issue 6**

87099 This utility is marked as part of the User Portability Utilities option.

87100 The APPLICATION USAGE section is added.

87101 The obsolescent `-r` option is removed.87102 The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had
87103 incorrectly been marked LEGACY in Issue 5.87104 The `-H` and `-L` options for symbolic links are added as described in the IEEE P1003.2b draft
87105 standard.87106 **Issue 7**87107 The *du* utility is moved from the User Portability Utilities option to the Base. User Portability
87108 Utilities is now an option for interactive utilities.

87109 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

87110 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0089 [527] is applied.

87111 **NAME**

87112 echo — write arguments to standard output

87113 **SYNOPSIS**87114 echo [*string*...]87115 **DESCRIPTION**87116 The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are
87117 no arguments, only the <newline> is written.87118 **OPTIONS**87119 The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10
87120 of XBD [Section 12.2](#) (on page 216); "--" shall be recognized as a string operand.

87121 Implementations shall not support any options.

87122 **OPERANDS**

87123 The following operands shall be supported:

87124 *string* A string to be written to standard output. If the first operand is *-n*, or if any of the
87125 operands contain a <backslash> character, the results are implementation-defined.87126 XSI On XSI-conformant systems, if the first operand is *-n*, it shall be treated as a string,
87127 not an option. The following character sequences shall be recognized on XSI-
87128 conformant systems within any of the arguments:

87129	<code>\a</code>	Write an <alert>.
87130	<code>\b</code>	Write a <backspace>.
87131	<code>\c</code>	Suppress the <newline> that otherwise follows the final argument in the 87132 output. All characters following the ' <code>\c</code> ' in the arguments shall be 87133 ignored.
87134	<code>\f</code>	Write a <form-feed>.
87135	<code>\n</code>	Write a <newline>.
87136	<code>\r</code>	Write a <carriage-return>.
87137	<code>\t</code>	Write a <tab>.
87138	<code>\v</code>	Write a <vertical-tab>.
87139	<code>\\</code>	Write a <backslash> character.
87140	<code>\0num</code>	Write an 8-bit value that is the zero, one, two, or three-digit octal number 87141 <i>num</i> .

87142 **STDIN**

87143 Not used.

87144 **INPUT FILES**

87145 None.

87146 **ENVIRONMENT VARIABLES**87147 The following environment variables shall affect the execution of *echo*:87148 *LANG* Provide a default value for the internationalization variables that are unset or null.
87149 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
87150 variables used to determine the values of locale categories.)

87151 `LC_ALL` If set to a non-empty string value, override the values of all the other
87152 internationalization variables.

87153 XSI `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
87154 characters (for example, single-byte as opposed to multi-byte characters in
87155 arguments).

87156 `LC_MESSAGES`
87157 Determine the locale that should be used to affect the format and contents of
87158 diagnostic messages written to standard error.

87159 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

87160 ASYNCHRONOUS EVENTS

87161 Default.

87162 STDOUT

87163 The *echo* utility arguments shall be separated by single <space> characters and a <newline>
87164 XSI character shall follow the last argument. Output transformations shall occur based on the
87165 escape sequences in the input. See the OPERANDS section.

87166 STDERR

87167 The standard error shall be used only for diagnostic messages.

87168 OUTPUT FILES

87169 None.

87170 EXTENDED DESCRIPTION

87171 None.

87172 EXIT STATUS

87173 The following exit values shall be returned:

87174 0 Successful completion.

87175 >0 An error occurred.

87176 CONSEQUENCES OF ERRORS

87177 Default.

87178 APPLICATION USAGE

87179 It is not possible to use *echo* portably across all POSIX systems unless both `-n` (as the first
87180 argument) and escape sequences are omitted.

87181 The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo*
87182 utility as follows (assuming that *IFS* has its standard value or is unset):

87183 The historic System V *echo* and the requirements on XSI implementations in this volume of
87184 POSIX.1-2017 are equivalent to:

```
87185 printf "%b\n" "$*"

```

87186 The BSD *echo* is equivalent to:

```
87187 if [ "X$1" = "X-n" ]
87188 then
87189     shift
87190     printf "%s" "$*"
87191 else
87192     printf "%s\n" "$*"
87193 fi

```

87194 New applications are encouraged to use *printf* instead of *echo*.

87195 **EXAMPLES**

87196 None.

87197 **RATIONALE**

87198 The *echo* utility has not been made obsolescent because of its extremely widespread use in
87199 historical applications. Conforming applications that wish to do prompting without <newline>
87200 characters or that could possibly be expecting to echo a **-n**, should use the *printf* utility derived
87201 from the Ninth Edition system.

87202 As specified, *echo* writes its arguments in the simplest of ways. The two different historical
87203 versions of *echo* vary in fatally incompatible ways.

87204 The BSD *echo* checks the first argument for the string **-n** which causes it to suppress the
87205 <newline> that would otherwise follow the final argument in the output.

87206 The System V *echo* does not support any options, but allows escape sequences within its
87207 operands, as described for XSI implementations in the OPERANDS section.

87208 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications
87209 depend on *echo* to echo *all* of its arguments, except for the **-n** option in the BSD version.

87210 **FUTURE DIRECTIONS**

87211 None.

87212 **SEE ALSO**

87213 *printf*

87214 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

87215 **CHANGE HISTORY**

87216 First released in Issue 2.

87217 **Issue 5**

87218 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not”
87219 support any options; in the previous issue this said “need not”.

87220 **Issue 6**

87221 The following new requirements on POSIX implementations derive from alignment with the
87222 Single UNIX Specification:

87223 A set of character sequences is defined as *string* operands.

87224 *LC_CTYPE* is added to the list of environment variables affecting *echo*.

87225 In the OPTIONS section, implementations shall not support any options.

87226 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can
87227 accommodate historical BSD behavior.

87228 **Issue 7**

87229 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

87230 **NAME**

87231 ed ‡edit text

87232 **SYNOPSIS**87233 ed [-p *string*] [-s] [*file*]87234 **DESCRIPTION**

87235 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In
 87236 command mode the input characters shall be interpreted as commands, and in input mode they
 87237 shall be interpreted as text. See the EXTENDED DESCRIPTION section.

87238 If an operand is '-', the results are unspecified.

87239 **OPTIONS**

87240 The *ed* utility shall conform to XBD [Section 12.2](#) (on page 216), except for the unspecified usage
 87241 of '-'.

87242 The following options shall be supported:

87243 **-p** *string* Use *string* as the prompt string when in command mode. By default, there shall be
 87244 no prompt string.

87245 **-s** Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'
 87246 prompt after a *!command*.

87247 **OPERANDS**

87248 The following operand shall be supported:

87249 *file* If the *file* argument is given, *ed* shall simulate an **e** command on the file named by
 87250 the pathname, *file*, before accepting commands from the standard input.

87251 **STDIN**

87252 The standard input shall be a text file consisting of commands, as described in the EXTENDED
 87253 DESCRIPTION section.

87254 **INPUT FILES**

87255 The input files shall be text files.

87256 **ENVIRONMENT VARIABLES**87257 The following environment variables shall affect the execution of *ed*:

87258 **HOME** Determine the pathname of the user's home directory.

87259 **LANG** Provide a default value for the internationalization variables that are unset or null.
 87260 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 87261 variables used to determine the values of locale categories.)

87262 **LC_ALL** If set to a non-empty string value, override the values of all the other
 87263 internationalization variables.

87264 **LC_COLLATE**

87265 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 87266 character collating elements within regular expressions.

87267 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 87268 characters (for example, single-byte as opposed to multi-byte characters in
 87269 arguments and input files) and the behavior of character classes within regular
 87270 expressions.

- 87271 *LC_MESSAGES*
- 87272 Determine the locale that should be used to affect the format and contents of
- 87273 diagnostic messages written to standard error and informative messages written to
- 87274 standard output.
- 87275 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 87276 **ASYNCHRONOUS EVENTS**
- 87277 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
- 87278 section in [Section 1.4](#), on page 2336) with the following exceptions:
- 87279 SIGINT The *ed* utility shall interrupt its current activity, write the string "?\n" to standard
- 87280 output, and return to command mode (see the EXTENDED DESCRIPTION
- 87281 section).
- 87282 SIGHUP If the buffer is not empty and has changed since the last write, the *ed* utility shall
- 87283 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the
- 87284 current directory shall be used; if that fails, the file named **ed.hup** in the directory
- 87285 named by the *HOME* environment variable shall be used. In any case, the *ed* utility
- 87286 shall exit without writing the file to the currently remembered pathname and
- 87287 without returning to command mode.
- 87288 SIGQUIT The *ed* utility shall ignore this event.
- 87289 **STDOUT**
- 87290 Various editing commands and the prompting feature (see **-p**) write to standard output, as
- 87291 described in the EXTENDED DESCRIPTION section.
- 87292 **STDERR**
- 87293 The standard error shall be used only for diagnostic messages.
- 87294 **OUTPUT FILES**
- 87295 The output files shall be text files whose formats are dependent on the editing commands given.
- 87296 **EXTENDED DESCRIPTION**
- 87297 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
- 87298 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.
- 87299 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a
- 87300 single-character *command*, possibly followed by parameters to that command. These addresses
- 87301 specify one or more lines in the buffer. Every command that requires addresses has default
- 87302 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the
- 87303 prompt string shall be written to standard output before each command is read.
- 87304 In general, only one command can appear on a line. Certain commands allow text to be input.
- 87305 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be
- 87306 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.
- 87307 Input mode is terminated by entering a line consisting of two characters: a <period> ('.')
- 87308 followed by a <newline>. This line is not considered part of the input text.

87309 Regular Expressions in ed

87310 The *ed* utility shall support basic regular expressions, as described in XBD [Section 9.3](#) (on page
87311 183). Since regular expressions in *ed* are always matched against single lines (excluding the
87312 terminating <newline> characters), never against any larger section of text, there is no way for a
87313 regular expression to match a <newline>.

87314 A null RE shall be equivalent to the last RE encountered.

87315 Regular expressions are used in addresses to specify lines, and in some commands (for example,
87316 the **s** substitute command) to specify portions of a line to be substituted.

87317 Addresses in ed

87318 Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a
87319 command. The current line number is the address of the current line. If the edit buffer is not
87320 empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

87321 Addresses shall be constructed as follows:

- 87322 1. The <period> character ('.') shall address the current line.
- 87323 2. The <dollar-sign> character ('\$ ') shall address the last line of the edit buffer.
- 87324 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 87325 4. The <apostrophe>-*x* character pair ("'*x*'") shall address the line marked with the mark
87326 name character *x*, which shall be a lowercase letter from the portable character set. It shall
87327 be an error if the character has not been set to mark a line or if the line that was marked is
87328 not currently present in the edit buffer.
- 87329 5. A BRE enclosed by <slash> characters ('/') shall address the first line found by
87330 searching forwards from the line following the current line toward the end of the edit
87331 buffer and stopping at the first line for which the line excluding the terminating
87332 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of
87333 <slash> characters shall address the next line for which the line excluding the terminating
87334 <newline> matches the last BRE encountered. In addition, the second <slash> can be
87335 omitted at the end of a command line. Within the BRE, a <backslash>-<slash> pair ("\/")
87336 shall represent a literal <slash> instead of the BRE delimiter. If necessary, the search shall
87337 wrap around to the beginning of the buffer and continue up to and including the current
87338 line, so that the entire buffer is searched.
- 87339 6. A BRE enclosed by <question-mark> characters ('?') shall address the first line found by
87340 searching backwards from the line preceding the current line toward the beginning of the
87341 edit buffer and stopping at the first line for which the line excluding the terminating
87342 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of
87343 <question-mark> characters ("??") shall address the previous line for which the line
87344 excluding the terminating <newline> matches the last BRE encountered. In addition, the
87345 second <question-mark> can be omitted at the end of a command line. Within the BRE, a
87346 <backslash>-<question-mark> pair ("\?") shall represent a literal <question-mark>
87347 instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the
87348 buffer and continue up to and including the current line, so that the entire buffer is
87349 searched.
- 87350 7. A <plus-sign> ('+') or <hyphen-minus> character ('-') followed by a decimal number
87351 shall address the current line plus or minus the number. A <plus-sign> or <hyphen-
87352 minus> character not followed by a decimal number shall address the current line plus or
87353 minus 1.

87354 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.
87355 Address offsets are constructed as follows:

87356 A <plus-sign> or <hyphen-minus> character followed by a decimal number shall add or
87357 subtract, respectively, the indicated number of lines to or from the address. A <plus-sign>
87358 or <hyphen-minus> character not followed by a decimal number shall add or subtract 1 to
87359 or from the address.

87360 A decimal number shall add the indicated number of lines to the address.

87361 It shall not be an error for an intermediate address value to be less than zero or greater than the
87362 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
87363 greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a
87364 matching line.

87365 Commands accept zero, one, or two addresses. If more than the required number of addresses
87366 are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more
87367 than the required number of addresses are provided to a command, the addresses specified first
87368 shall be evaluated and then discarded until the maximum number of valid addresses remain, for
87369 the specified command.

87370 Addresses shall be separated from each other by a <comma> (',') or <semicolon> character
87371 (';'). In the case of a <semicolon> separator, the current line ('.') shall be set to the first
87372 address, and only then will the second address be calculated. This feature can be used to
87373 determine the starting line for forwards and backwards searches; see rules 5. and 6.

87374 Addresses can be omitted on either side of the <comma> or <semicolon> separator, in which
87375 case the resulting address pairs shall be as follows:

Specified	Resulting
,	1 , \$
, addr	1 , addr
addr ,	addr , addr
;	. ; \$
; addr	. ; addr
addr ;	addr ; addr

87383 Any <blank> characters included between addresses, address separators, or address offsets shall
87384 be ignored.

87385 **Commands in ed**

87386 In the following list of *ed* commands, the default addresses are shown in parentheses. The
87387 number of addresses shown in the default shall be the number expected by the command. The
87388 parentheses are not part of the address; they show that the given addresses are the default.

87389 It is generally invalid for more than one command to appear on a line. However, any command
87390 (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for
87391 the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be
87392 written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used
87393 with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but
87394 it is unspecified whether the suffix writes the current line again in the requested format or
87395 whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l**
87396 suffix) shall either write just the current line or write it twice—once as specified for **p** and once
87397 as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

87398 Each address component can be preceded by zero or more <blank> characters. The command

87399 letter can be preceded by zero or more <blank> characters. If a suffix letter (**l**, **n**, or **p**) is given,
87400 the application shall ensure that it immediately follows the command.

87401 The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the
87402 command letter by one or more <blank> characters.

87403 If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed*
87404 shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.
87405 The *ed* utility shall write the string:

87406 "?\n"

87407 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to
87408 standard output and shall continue in command mode with the current line number unchanged.
87409 If the **e** or **q** command is repeated with no intervening command, it shall take effect.

87410 If a terminal disconnect (see XBD [Chapter 11](#) (on page 199), Modem Disconnect and Closing a
87411 Device Terminal), is detected:

87412 If accompanied by a SIGHUP signal, the *ed* utility shall operate as described in the
87413 ASYNCHRONOUS EVENTS section for a SIGHUP signal.

87414 If not accompanied by a SIGHUP signal, the *ed* utility shall act as if an end-of-file had been
87415 detected on standard input.

87416 If an end-of-file is detected on standard input:

87417 If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command
87418 mode. It is unspecified if any partially entered lines (that is, input text without a
87419 terminating <newline>) are discarded from the input text.

87420 If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

87421 If the closing delimiter of an RE or of a replacement string (for example, '/') in a **g**, **G**, **s**, **v**, or **V**
87422 command would be the last character before a <newline>, that delimiter can be omitted, in
87423 which case the addressed line shall be written. For example, the following pairs of commands
87424 are equivalent:

87425 *s/s1/s2* *s/s1/s2/p*
87426 *g/s1* *g/s1/p*
87427 *?s1* *?s1?*

87428 If an invalid command is entered, *ed* shall write the string:

87429 "?\n"

87430 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to
87431 standard output and shall continue in command mode with the current line number unchanged.

87432 **Append Command**

87433 *Synopsis:* (**.**)**a**
87434 <*text*>
87435 .

87436 The **a** command shall read the given text and append it after the addressed line; the current line
87437 number shall become the address of the last inserted line or, if there were none, the addressed
87438 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at
87439 the beginning of the buffer.

87440 **Change Command**

87441 *Synopsis:* (. , .)c
 87442 <text>
 87443 .

87444 The **c** command shall delete the addressed lines, then accept input text that replaces these lines;
 87445 the current line shall be set to the address of the last line input; or, if there were none, at the line
 87446 after the last line deleted; if the lines deleted were originally at the end of the buffer, the current
 87447 line number shall be set to the address of the new last line; if no lines remain in the buffer, the
 87448 current line number shall be set to zero. Address 0 shall be valid for this command; it shall be
 87449 interpreted as if address 1 were specified.

87450 **Delete Command**

87451 *Synopsis:* (. , .)d

87452 The **d** command shall delete the addressed lines from the buffer. The address of the line after the
 87453 last line deleted shall become the current line number; if the lines deleted were originally at the
 87454 end of the buffer, the current line number shall be set to the address of the new last line; if no
 87455 lines remain in the buffer, the current line number shall be set to zero.

87456 **Edit Command**

87457 *Synopsis:* e [*file*]

87458 The **e** command shall delete the entire contents of the buffer and then read in the file named by
 87459 the pathname *file*. The current line number shall be set to the address of the last line of the
 87460 buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see
 87461 the **f** command). The number of bytes read shall be written to standard output, unless the **-s**
 87462 option was specified, in the following format:

87463 "%d\n", <number of bytes read>

87464 The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**,
 87465 and **w** commands. If *file* is replaced by '!', the rest of the line shall be taken to be a shell
 87466 command line whose output is to be read. Such a shell command line shall not be remembered
 87467 as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.
 87468 If the buffer has changed since the last time the entire buffer was written, the user shall be
 87469 warned, as described previously.

87470 **Edit Without Checking Command**

87471 *Synopsis:* E [*file*]

87472 The **E** command shall possess all properties and restrictions of the **e** command except that the
 87473 editor shall not check to see whether any changes have been made to the buffer since the last **w**
 87474 command.

87475 Filename Command

87476 *Synopsis:* *f* [*file*]

87477 If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether
87478 the name is changed or not, it shall then write the (possibly new) currently remembered
87479 pathname to the standard output in the following format:

87480 "%s\n", <pathname>

87481 The current line number shall be unchanged.

87482 Global Command

87483 *Synopsis:* (1,\$)g/RE/command list

87484 In the **g** command, the first step shall be to mark every line for which the line excluding the
87485 terminating <newline> matches the given RE. Then, going sequentially from the beginning of
87486 the file to the end of the file, the given *command list* shall be executed for each marked line, with
87487 the current line number set to the address of that line. Any line modified by the *command list*
87488 shall be unmarked. When the **g** command completes, the current line number shall have the
87489 value assigned by the last command in the *command list*. If there were no matching lines, the
87490 current line number shall not be changed. A single command or the first of a list of commands
87491 shall appear on the same line as the global command. All lines of a multi-line list except the last
87492 line shall be ended with a <backslash> preceding the terminating <newline>; the **a**, **i**, and **c**
87493 commands and associated input are permitted. The '.' terminating input mode can be omitted
87494 if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p**
87495 command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined
87496 results. Any character other than <space> or <newline> can be used instead of a <slash> to
87497 delimit the RE. Within the RE, the RE delimiter itself can be used as a literal character if it is
87498 preceded by a <backslash>.

87499 Interactive Global Command

87500 *Synopsis:* (1,\$)G/RE/

87501 In the **G** command, the first step shall be to mark every line for which the line excluding the
87502 terminating <newline> matches the given RE. Then, for every such line, that line shall be
87503 written, the current line number shall be set to the address of that line, and any one command
87504 (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline>
87505 shall act as a null command (causing no action to be taken on the current line); an '&' shall
87506 cause the re-execution of the most recent non-null command executed within the current
87507 invocation of **G**. Note that the commands input as part of the execution of the **G** command can
87508 address and affect any lines in the buffer. Any line modified by the command shall be
87509 unmarked. The final value of the current line number shall be the value set by the last command
87510 successfully executed. (Note that the last command successfully executed shall be the **G**
87511 command itself if a command fails or the null command is specified.) If there were no matching
87512 lines, the current line number shall not be changed. The **G** command can be terminated by a
87513 SIGINT signal. Any character other than <space> or <newline> can be used instead of a <slash>
87514 to delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a
87515 literal character if it is preceded by a <backslash>.

87516 **Help Command**87517 *Synopsis:* h87518 The **h** command shall write a short message to standard output that explains the reason for the
87519 most recent '?' notification. The current line number shall be unchanged.87520 **Help-Mode Command**87521 *Synopsis:* H87522 The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command)
87523 shall be written to standard output for all subsequent '?' notifications. The **H** command
87524 alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on,
87525 the **H** command also explains the previous '?' notification, if there was one. The current line
87526 number shall be unchanged.87527 **Insert Command**87528 *Synopsis:* (.) i
87529 <text>
87530 .87531 The **i** command shall insert the given text before the addressed line; the current line is set to the
87532 last inserted line or, if there was none, to the addressed line. This command differs from the **a**
87533 command only in the placement of the input text. Address 0 shall be valid for this command; it
87534 shall be interpreted as if address 1 were specified.87535 **Join Command**87536 *Synopsis:* (. , . +1) j87537 The **j** command shall join contiguous lines by removing the appropriate <newline> characters. If
87538 exactly one address is given, this command shall do nothing. If lines are joined, the current line
87539 number shall be set to the address of the joined line; otherwise, the current line number shall be
87540 unchanged.87541 **Mark Command**87542 *Synopsis:* (.) k x87543 The **k** command shall mark the addressed line with name *x*, which the application shall ensure
87544 is a lowercase letter from the portable character set. The address "'x" shall then refer to this
87545 line; the current line number shall be unchanged.87546 **List Command**87547 *Synopsis:* (. , .) l87548 The **l** command shall write to standard output the addressed lines in a visually unambiguous
87549 form. The characters listed in XBD [Table 5-1](#) (on page 121) ('\ ', '\a', '\b', '\f', '\r',
87550 '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not
87551 applicable. Non-printable characters not in the table shall be written as one three-digit octal
87552 number (with a preceding <backslash> character) for each byte in the character (most significant
87553 byte first).87554 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a
87555 <backslash>; the length at which folding occurs is unspecified, but should be appropriate for the

87556 output device. The end of each line shall be marked with a '\$', and '\$' characters within the
87557 text shall be written with a preceding <backslash>. An I command can be appended to any
87558 other command other than e, E, f, q, Q, r, w, or !. The current line number shall be set to the
87559 address of the last line written.

87560 **Move Command**

87561 *Synopsis:* (. , .) *address*

87562 The m command shall reposition the addressed lines after the line addressed by *address*.
87563 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning
87564 of the buffer. It shall be an error if address *address* falls within the range of moved lines. The
87565 current line number shall be set to the address of the last line moved.

87566 **Number Command**

87567 *Synopsis:* (. , .) n

87568 The n command shall write to standard output the addressed lines, preceding each line by its
87569 line number and a <tab>; the current line number shall be set to the address of the last line
87570 written. The n command can be appended to any command other than e, E, f, q, Q, r, w, or !.

87571 **Print Command**

87572 *Synopsis:* (. , .) p

87573 The p command shall write to standard output the addressed lines; the current line number shall
87574 be set to the address of the last line written. The p command can be appended to any command
87575 other than e, E, f, q, Q, r, w, or !.

87576 **Prompt Command**

87577 *Synopsis:* P

87578 The P command shall cause *ed* to prompt with an <asterisk> ('*') (or *string*, if -p is specified)
87579 for all subsequent commands. The P command alternatively shall turn this mode on and off; it
87580 shall be initially on if the -p option is specified; otherwise, off. The current line number shall be
87581 unchanged.

87582 **Quit Command**

87583 *Synopsis:* q

87584 The q command shall cause *ed* to exit. If the buffer has changed since the last time the entire
87585 buffer was written, the user shall be warned, as described previously.

87586 **Quit Without Checking Command**

87587 *Synopsis:* Q

87588 The Q command shall cause *ed* to exit without checking whether changes have been made in the
87589 buffer since the last w command.

87590 **Read Command**87591 *Synopsis:* (\$)**r** [*file*]

87592 The **r** command shall read in the file named by the pathname *file* and append it after the
 87593 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be
 87594 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed
 87595 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file
 87596 to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the
 87597 number of bytes read shall be written to standard output in the following format:

87598 "%d\n", <number of bytes read>

87599 The current line number shall be set to the address of the last line read in. If *file* is replaced by
 87600 '!', the rest of the line shall be taken to be a shell command line whose output is to be read.
 87601 Such a shell command line shall not be remembered as the current pathname.

87602 **Substitute Command**87603 *Synopsis:* (.,.)**s**/*RE/replacement/flags*

87604 The **s** command shall search each addressed line for an occurrence of the specified RE and
 87605 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the
 87606 following description of the **g** suffix. It is an error if the substitution fails on every addressed
 87607 line. Any character other than <space> or <newline> can be used instead of a <slash> to delimit
 87608 the RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal
 87609 character if it is preceded by a <backslash>. The current line shall be set to the address of the
 87610 last line on which a substitution occurred.

87611 An <ampersand> ('&') appearing in the *replacement* shall be replaced by the string matching the
 87612 RE on the current line. The special meaning of '&' in this context can be suppressed by
 87613 preceding it by <backslash>. As a more general feature, the characters '\n', where *n* is a digit,
 87614 shall be replaced by the text matched by the corresponding back-reference expression. If the
 87615 corresponding back-reference expression does not match, then the characters '\n' shall be
 87616 replaced by the empty string. When the character '%' is the only character in the *replacement*, the
 87617 *replacement* used in the most recent substitute command shall be used as the *replacement* in the
 87618 current substitute command; if there was no previous substitute command, the use of '%' in this
 87619 manner shall be an error. The '%' shall lose its special meaning when it is in a replacement
 87620 string of more than one character or is preceded by a <backslash>. For each <backslash>
 87621 encountered in scanning *replacement* from beginning to end, the following character shall lose its
 87622 special meaning (if any). It is unspecified what special meaning is given to any character other
 87623 than <backslash>, '&', '%', or digits.

87624 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the
 87625 <newline> in the *replacement* by preceding it by <backslash>. Such substitution cannot be done
 87626 as part of a **g** or **v** *command list*. The current line number shall be set to the address of the last
 87627 line on which a substitution is performed. If no substitution is performed, the current line
 87628 number shall be unchanged. If a line is split, a substitution shall be considered to have been
 87629 performed on each of the new lines for the purpose of determining the new current line number.
 87630 A substitution shall be considered to have been performed even if the replacement string is
 87631 identical to the string that it replaces.

87632 The application shall ensure that the value of *flags* is zero or more of:87633 *count* Substitute for the *count*th occurrence only of the RE found on each addressed line.

- 87634 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first
87635 one. If both **g** and *count* are specified, the results are unspecified.
- 87636 **l** Write to standard output the final line in which a substitution was made. The line shall
87637 be written in the format specified for the **l** command.
- 87638 **n** Write to standard output the final line in which a substitution was made. The line shall
87639 be written in the format specified for the **n** command.
- 87640 **p** Write to standard output the final line in which a substitution was made. The line shall
87641 be written in the format specified for the **p** command.

87642 **Copy Command**

87643 *Synopsis:* (, .) *taddress*

87644 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines
87645 shall be placed after address *address* (which can be 0); the current line number shall be set to the
87646 address of the last line added.

87647 **Undo Command**

87648 *Synopsis:* u

87649 The **u** command shall nullify the effect of the most recent command that modified anything in
87650 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes
87651 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no
87652 changes were made by the global command (such as with **g/RE/p**), the **u** command shall have
87653 no effect. The current line number shall be set to the value it had immediately before the
87654 command being undone started.

87655 **Global Non-Matched Command**

87656 *Synopsis:* (1 , \$) *v/RE/command list*

87657 This command shall be equivalent to the global command **g** except that the lines that are marked
87658 during the first step shall be those for which the line excluding the terminating <newline> does
87659 not match the RE.

87660 **Interactive Global Not-Matched Command**

87661 *Synopsis:* (1 , \$) *V/RE/*

87662 This command shall be equivalent to the interactive global command **G** except that the lines that
87663 are marked during the first step shall be those for which the line excluding the terminating
87664 <newline> does not match the RE.

87665 **Write Command**

87666 *Synopsis:* (1 , \$) *w [file]*

87667 The **w** command shall write the addressed lines into the file named by the pathname *file*. The
87668 command shall create the file, if it does not exist, or shall replace the contents of the existing file.
87669 The currently remembered pathname shall not be changed unless there is no remembered
87670 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used
87671 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is
87672 successful, the number of bytes written shall be written to standard output, unless the **-s** option
87673 was specified, in the following format:

87674 "%d\n", <number of bytes written>

87675 If *file* begins with '!', the rest of the line shall be taken to be a shell command line whose
 87676 standard input shall be the addressed lines. Such a shell command line shall not be remembered
 87677 as the current pathname. This usage of the write command with '!' shall not be considered as a
 87678 "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall
 87679 not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or
 87680 **q** commands.

87681 **Line Number Command**

87682 *Synopsis:* (\$)=

87683 The line number of the addressed line shall be written to standard output in the following
 87684 format:

87685 "%d\n", <line number>

87686 The current line number shall be unchanged by this command.

87687 **Shell Escape Command**

87688 *Synopsis:* !*command*

87689 The remainder of the line after the '!' shall be sent to the command interpreter to be
 87690 interpreted as a shell command line. Within the text of that shell command line, the unescaped
 87691 character '%' shall be replaced with the remembered pathname; if a '!' appears as the first
 87692 character of the command, it shall be replaced with the text of the previous shell command
 87693 executed via '!'. Thus, "!!" shall repeat the previous !*command*. If any replacements of '%' or
 87694 '!' are performed, the modified line shall be written to the standard output before *command* is
 87695 executed. The ! command shall write:

87696 "!\n"

87697 to standard output upon completion, unless the **-s** option is specified. The current line number
 87698 shall be unchanged.

87699 **Null Command**

87700 *Synopsis:* (.+1)

87701 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall
 87702 be equivalent to "+1p". The current line number shall be set to the address of the written line.

87703 **EXIT STATUS**

87704 The following exit values shall be returned:

87705 0 Successful completion without any file or command errors.

87706 >0 An error occurred.

87707 **CONSEQUENCES OF ERRORS**

87708 When an error in the input script is encountered, or when an error is detected that is a
 87709 consequence of the data (not) present in the file or due to an external condition such as a read or
 87710 write error:

87711 If the standard input is a terminal device file, all input shall be flushed, and a new
 87712 command read.

87713 If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

87714 APPLICATION USAGE

87715 Because of the extremely terse nature of the default error messages, the prudent script writer
87716 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some
87717 clue as to the cause is made available.

87718 In earlier versions of this standard, an obsolescent `-` option was described. This is no longer
87719 specified. Applications should use the `-s` option. Using `-` as a *file* operand now produces
87720 unspecified results. This allows implementations to continue to support the former required
87721 behavior.

87722 EXAMPLES

87723 None.

87724 RATIONALE

87725 The initial description of this utility was adapted from the SVID. It contains some features not
87726 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and
87727 BSD *ed* utilities include, but need not be limited to:

87728 The BSD `-` option does not suppress the `!'` prompt after a `!` command.

87729 BSD does not support the special meanings of the `'%` and `!'` characters within a `!`
87730 command.

87731 BSD does not support the *addresses* `';` and `' , '`.

87732 BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this
87733 volume of POSIX.1-2017.

87734 BSD does not support the `!'` character part of the **e**, **r**, or **w** commands.

87735 A failed **g** command in BSD sets the line number to the last line searched if there are no
87736 matches.

87737 BSD does not default the *command list* to the **p** command.

87738 BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.

87739 On BSD, if there is no inserted text, the insert command changes the current line to the
87740 referenced line `-1`; that is, the line before the specified line.

87741 On BSD, the *join* command with only a single address changes the current line to that
87742 address.

87743 BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p**
87744 command.

87745 BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.

87746 The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in
87747 this volume of POSIX.1-2017.

87748 The `-s` option was added to allow the functionality of the removed `-` option in a manner
87749 compatible with the Utility Syntax Guidelines.

87750 In early proposals there was a limit, `{ED_FILE_MAX}`, that described the historical limitations of
87751 some *ed* utilities in their handling of large files; some of these have had problems with files
87752 larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a
87753 *split* command in this volume of POSIX.1-2017. Since this limit was removed, this volume of
87754 POSIX.1-2017 requires that implementations document the file size limits imposed by *ed* in the

87755 conformance document. The limit {ED_LINE_MAX} was also removed; therefore, the global
87756 limit {LINE_MAX} is used for input and output lines.

87757 The manner in which the **I** command writes non-printable characters was changed to avoid the
87758 historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous
87759 because most terminals simply replace overstruck characters, making the **I** format not useful for
87760 its intended purpose of unambiguously understanding the content of the line. The historical
87761 <backslash>-escapes were also ambiguous. (The string "a\0011" could represent a line
87762 containing those six characters or a line containing the three characters 'a', a byte with a binary
87763 value of 1, and a 1.) In the format required here, a <backslash> appearing in the line is written as
87764 "\\\" so that the output is truly unambiguous. The method of marking the ends of lines was
87765 adopted from the *ex* editor and is required for any line ending in <space> characters; the '\$' is
87766 placed on all lines so that a real '\$' at the end of a line cannot be misinterpreted.

87767 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
87768 but this has been modified in this version.

87769 The description of how a NUL is written was removed. The NUL character cannot be in text
87770 files, and this volume of POSIX.1-2017 should not dictate behavior in the case of undefined,
87771 erroneous input.

87772 Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands
87773 are not patterns.

87774 Early proposals stated that the **-p** option worked only when standard input was associated with
87775 a terminal device. This has been changed to conform to historical implementations, thereby
87776 allowing applications to interpose themselves between a user and the *ed* utility.

87777 The form of the substitute command that uses the **n** suffix was limited in some historical
87778 documentation (where this was described incorrectly as "backreferencing"). This limit has been
87779 omitted because there is no reason why an editor processing lines of {LINE_MAX} length should
87780 have this restriction. The command **s/x/X/2047** should be able to substitute the 2047th occurrence
87781 of 'x' on a line.

87782 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made
87783 unspecified because BSD-based systems allow this, whereas System V does not.

87784 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file
87785 have been deleted. Since this volume of POSIX.1-2017 refers to the **q** command in this instance,
87786 such behavior is not allowed.

87787 Some historical implementations returned exit status zero even if command errors had occurred;
87788 this is not allowed by this volume of POSIX.1-2017.

87789 Some historical implementations contained a bug that allowed a single <period> to be entered in
87790 input mode as <backslash> <period> <newline>. This is not allowed by *ed* because there is no
87791 description of escaping any of the characters in input mode; <backslash> characters are entered
87792 into the buffer exactly as typed. The typical method of entering a single <period> has been to
87793 precede it with another character and then use the substitute command to delete that character.

87794 It is difficult under some modes of some versions of historical operating system terminal drivers
87795 to distinguish between an end-of-file condition and terminal disconnect. POSIX.1-2017 does not
87796 require implementations to distinguish between the two situations, which permits historical
87797 implementations of the *ed* utility on historical platforms to conform. Implementations are
87798 encouraged to distinguish between the two, if possible, and take appropriate action on terminal
87799 disconnect.

87800 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the

87801 start of the edit buffer. When the buffer was empty the command `.=` returned zero. POSIX.1-2017
87802 requires conformance to historical practice.

87803 For consistency with the `a` and `r` commands and better user functionality, the `i` and `c` commands
87804 must also accept an address of 0, in which case `0i` is treated as `1i` and likewise for the `c`
87805 command.

87806 All of the following are valid addresses:

87807 `+++` Three lines after the current line.
87808 `/pattern/-` One line before the next occurrence of pattern.
87809 `-2` Two lines before the current line.
87810 `3 ---- 2` Line one (note the intermediate negative address).
87811 `1 2 3` Line six.

87812 Any number of addresses can be provided to commands taking addresses; for example,
87813 `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses
87814 accepted by the `print` command. This, in combination with the `<semicolon>` delimiter, permits
87815 users to create commands based on ordered patterns in the file. For example, the command
87816 `"3;/foo/;+2p"` will display the first line after line 3 that contains the pattern `foo`, plus the next
87817 two lines. Note that the address `"3;"` must still be evaluated before being discarded, because
87818 the search origin for the `"/foo/"` command depends on this.

87819 Historically, `ed` disallowed address chains, as discussed above, consisting solely of `<comma>` or
87820 `<semicolon>` separators; for example, `","` or `";;"` were considered an error. For
87821 consistency of address specification, this restriction is removed. The following table lists some of
87822 the address forms now possible:

Address	Addr1	Addr2	Status	Comment
7,	7	7	Historical	Valid, but erroneous.
7,5,	5	5	Historical	
7,5,9	5	9	Historical	
7,9	7	9	Historical	
7,+	7	8	Historical	
,	1	\$	Historical	
,7	1	7	Extension	
,,	\$	\$	Extension	
,;	\$	\$	Extension	
7;	7	7	Historical	
7;5;	5	5	Historical	
7;5;9	5	9	Historical	
7;5,9	5	9	Historical	
7;\$;4	\$	4	Historical	
7;9	7	9	Historical	
7;+	7	8	Historical	
;	.	\$	Historical	
;7	.	7	Extension	
;;	\$	\$	Extension	
;;	\$	\$	Extension	

87844 Historically, `ed` accepted the `'^'` character as an address, in which case it was identical to the
87845 `<hyphen-minus>` character. POSIX.1-2017 does not require or prohibit this behavior.

87846 **FUTURE DIRECTIONS**

87847 None.

87848 **SEE ALSO**87849 [Section 1.4](#) (on page 2336), *ex*, *sed*, *sh*, *vi*87850 [XBD Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 216)87852 **CHANGE HISTORY**

87853 First released in Issue 2.

87854 **Issue 5**87855 In the OPTIONS section, the meaning of `-s` and `-` is clarified.

87856 A second FUTURE DIRECTION is added.

87857 **Issue 6**

87858 The obsolescent single-minus form is removed.

87859 A second APPLICATION USAGE note is added.

87860 The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

87861 The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when end-of-file is detected and when terminal disconnect is detected.

87862 The normative text is reworded to avoid use of the term “must” for application requirements.

87863 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: “Any line modified by the *command list* shall be unmarked.” to the **G** command. This change corresponds to a similar change made to the **g** command in the first version of this standard.

87864 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/7 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

87865 **Issue 7**87866 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is `'_'`.

87867 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

87868 SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

87869 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

87870 SD5-XCU-ERN-135 is applied, removing some RATIONALE text that is no longer applicable.

87871 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0090 [584], XCU/TC2-2008/0091 [584], and XCU/TC2-2008/0092 [584] are applied.

87880 **NAME**87881 `env` — set the environment for command invocation87882 **SYNOPSIS**87883 `env [-i] [name=value]... [utility [argument...]]`87884 **DESCRIPTION**87885 The `env` utility shall obtain the current environment, modify it according to its arguments, then
87886 invoke the utility named by the `utility` operand with the modified environment.87887 Optional arguments shall be passed to `utility`.87888 If no `utility` operand is specified, the resulting environment shall be written to the standard
87889 output, with one `name=value` pair per line.

87890 If the first argument is '-', the results are unspecified.

87891 **OPTIONS**87892 The `env` utility shall conform to XBD [Section 12.2](#) (on page 216), except for the unspecified usage
87893 of '-'.
87894 The following options shall be supported:

87894 The following options shall be supported:

87895 **-i** Invoke `utility` with exactly the environment specified by the arguments; the
87896 inherited environment shall be ignored completely.87897 **OPERANDS**

87898 The following operands shall be supported:

87899 `name=value` Arguments of the form `name=value` shall modify the execution environment, and
87900 shall be placed into the inherited environment before the `utility` is invoked.87901 `utility` The name of the utility to be invoked. If the `utility` operand names any of the
87902 special built-in utilities in [Section 2.14](#) (on page 2384), the results are undefined.87903 `argument` A string to pass as an argument for the invoked utility.87904 **STDIN**

87905 Not used.

87906 **INPUT FILES**

87907 None.

87908 **ENVIRONMENT VARIABLES**87909 The following environment variables shall affect the execution of `env`:87910 `LANG` Provide a default value for the internationalization variables that are unset or null.
87911 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
87912 variables used to determine the values of locale categories.)87913 `LC_ALL` If set to a non-empty string value, override the values of all the other
87914 internationalization variables.87915 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
87916 characters (for example, single-byte as opposed to multi-byte characters in
87917 arguments).87918 `LC_MESSAGES`87919 Determine the locale that should be used to affect the format and contents of
87920 diagnostic messages written to standard error.

- 87921 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 87922 **PATH** Determine the location of the *utility*, as described in XBD [Chapter 8](#) (on page 173).
87923 If *PATH* is specified as a *name=value* operand to *env*, the *value* given shall be used in
87924 the search for *utility*.
- 87925 **ASYNCHRONOUS EVENTS**
87926 Default.
- 87927 **STDOUT**
87928 If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be
87929 written in the form:
87930 "%s=%s\n", <name>, <value>
87931 If the *utility* operand is specified, the *env* utility shall not write to standard output.
- 87932 **STDERR**
87933 The standard error shall be used only for diagnostic messages.
- 87934 **OUTPUT FILES**
87935 None.
- 87936 **EXTENDED DESCRIPTION**
87937 None.
- 87938 **EXIT STATUS**
87939 If *utility* is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise, the *env*
87940 utility shall exit with one of the following values:
87941 0 The *env* utility completed successfully.
87942 1–125 An error occurred in the *env* utility.
87943 126 The utility specified by *utility* was found but could not be invoked.
87944 127 The utility specified by *utility* could not be found.
- 87945 **CONSEQUENCES OF ERRORS**
87946 Default.
- 87947 **APPLICATION USAGE**
87948 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
87949 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
87950 utility exited with an error indication”. The value 127 was chosen because it is not commonly
87951 used for other meanings; most utilities use small values for “normal error conditions” and the
87952 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
87953 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
87954 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
87955 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
87956 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
87957 any other reason.
87958 Historical implementations of the *env* utility use the *execvp()* or *execlp()* functions defined in the
87959 System Interfaces volume of POSIX.1-2017 to invoke the specified utility; this provides better
87960 performance and keeps users from having to escape characters with special meaning to the shell.
87961 Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are
87962 not found.

87963 **EXAMPLES**

87964 The following command:

87965 `env -i PATH=/mybin:"$PATH" $(getconf V7_ENV) mygrep xyz myfile`87966 invokes the command *mygrep* with a new *PATH* value as the only entry in its environment other
87967 than any variables required by the implementation for conformance. In this case, *PATH* is used
87968 to locate *mygrep*, which is expected to reside in **/mybin**.87969 **RATIONALE**87970 As with all other utilities that invoke other utilities, this volume of POSIX.1-2017 only specifies
87971 what *env* does with standard input, standard output, standard error, input files, and output files.
87972 If a utility is executed, it is not constrained by the specification of input and output by *env*.87973 The `-i` option was added to allow the functionality of the removed `-` option in a manner
87974 compatible with the Utility Syntax Guidelines. It is possible to create a non-conforming
87975 environment using the `-i` option, as it may remove environment variables required by the
87976 implementation for conformance. The following will preserve these environment variables as
87977 well as preserve the *PATH* for conforming utilities:

```

87978 IFS='
87979 '
87980 # The preceding value should be <space><tab><newline>.
87981 # Set IFS to its default value.

87982 set -f
87983 # disable pathname expansion

87984 \unalias -a
87985 # Unset all possible aliases.
87986 # Note that unalias is escaped to prevent an alias
87987 # being used for unalias.
87988 # This step is not strictly necessary, since aliases are not inherited,
87989 # and the ENV environment variable is only used by interactive shells,
87990 # the only way any aliases can exist in a script is if it defines them
87991 # itself.

87992 unset -f env getconf
87993 # Ensure env and getconf are not user functions.

87994 env -i $(getconf V7_ENV) PATH="$(getconf PATH)" command

```

87995 Some have suggested that *env* is redundant since the same effect is achieved by:87996 `name=value ... utility [argument ...]`87997 The example is equivalent to *env* when an environment variable is being added to the
87998 environment of the command, but not when the environment is being set to the given value.
87999 The *env* utility also writes out the current environment if invoked without arguments. There is
88000 sufficient functionality beyond what the example provides to justify inclusion of *env*.88001 **FUTURE DIRECTIONS**

88002 None.

88003 **SEE ALSO**88004 [Section 2.14](#) (on page 2384), [Section 2.5](#) (on page 2349)88005 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

88006 **CHANGE HISTORY**

88007 First released in Issue 2.

88008 **Issue 7**88009 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
88010 argument is '- '.88011 Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use
88012 the *env* utility to preserve a conforming environment.

88013 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88014 The EXAMPLES section is revised to change the use of *env -i*.

88015 **NAME**88016 ex \ddagger 'text editor88017 **SYNOPSIS**88018 UP ex [-rR] [-s|-v] [-c *command*] [-t *tagstring*] [-w *size*] [*file...*]88019 **DESCRIPTION**

88020 The *ex* utility is a line-oriented text editor. There are two other modes of the editor \ddagger open and
 88021 visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**
 88022 and **visual** commands and in *vi*.

88023 If an operand is '-', the results are unspecified.

88024 This section uses the term *edit buffer* to describe the current working text. No specific
 88025 implementation is implied by this term. All editing changes are performed on the edit buffer,
 88026 and no changes to it shall affect any file until an editor command writes the file.

88027 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,
 88028 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands
 88029 cannot be supported on such terminals, this condition shall not produce an error message such
 88030 as "not an editor command" or report a syntax error. The implementation may either accept the
 88031 commands and produce results on the screen that are the result of an unsuccessful attempt to
 88032 meet the requirements of this volume of POSIX.1-2017 or report an error describing the terminal-
 88033 related deficiency.

88034 **OPTIONS**

88035 The *ex* utility shall conform to XBD [Section 12.2](#) (on page 216), except for the unspecified usage
 88036 of '-', and that '+' may be recognized as an option delimiter as well as '-'.

88037 The following options shall be supported:

88038 **-c *command*** Specify an initial command to be executed in the first edit buffer loaded from an
 88039 existing file (see the EXTENDED DESCRIPTION section). Implementations may
 88040 support more than a single -c option. In such implementations, the specified
 88041 commands shall be executed in the order specified on the command line.

88042 **-r** Recover the named files (see the EXTENDED DESCRIPTION section). Recovery
 88043 information for a file shall be saved during an editor or system crash (for example,
 88044 when the editor is terminated by a signal which the editor can catch), or after the
 88045 use of an *ex* **preserve** command.

88046 A *crash* in this context is an unexpected failure of the system or utility that requires
 88047 restarting the failed system or utility. A system crash implies that any utilities
 88048 running at the time also crash. In the case of an editor or system crash, the number
 88049 of changes to the edit buffer (since the most recent **preserve** command) that will be
 88050 recovered is unspecified.

88051 If no *file* operands are given and the -t option is not specified, all other options, the
 88052 *EXINIT* variable, and any **.exrc** files shall be ignored; a list of all recoverable files
 88053 available to the invoking user shall be written, and the editor shall exit normally
 88054 without further action.

88055 **-R** Set **readonly** edit option.

88056 **-s** Prepare *ex* for batch use by taking the following actions:

- 88057 Suppress writing prompts and informational (but not diagnostic) messages.
- 88058 Ignore the value of *TERM* and any implementation default terminal type and
88059 assume the terminal is a type incapable of supporting open or visual modes;
88060 see the **visual** command and the description of *vi*.
- 88061 Suppress the use of the *EXINIT* environment variable and the reading of any
88062 **.exrc** file; see the EXTENDED DESCRIPTION section.
- 88063 Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 88064 **-t tagstring** Edit the file containing the specified *tagstring*; see *ctags*. The tags feature
88065 represented by **-t tagstring** and the **tag** command is optional. It shall be provided
88066 on any system that also provides a conforming implementation of *ctags*; otherwise,
88067 the use of **-t** produces undefined results. On any system, it shall be an error to
88068 specify more than a single **-t** option.
- 88069 **-v** Begin in visual mode (see *vi*).
- 88070 **-w size** Set the value of the *window* editor option to *size*.

88071 OPERANDS

88072 The following operand shall be supported:

88073 *file* A pathname of a file to be edited.

88074 STDIN

88075 The standard input consists of a series of commands and input text, as described in the
88076 EXTENDED DESCRIPTION section. The implementation may limit each line of standard input
88077 to a length of {LINE_MAX}.

88078 If the standard input is not a terminal device, it shall be as if the **-s** option had been specified.

88079 If a read from the standard input returns an error, or if the editor detects an end-of-file condition
88080 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

88081 INPUT FILES

88082 Input files shall be text files or files that would be text files except for an incomplete last line that
88083 is not longer than {LINE_MAX}-1 bytes in length and contains no NUL characters. By default,
88084 any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other
88085 forms of files may optionally be allowed by *ex* implementations.

88086 The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED
88087 DESCRIPTION section.

88088 By default, the editor shall read lines from the files to be edited without interpreting any of those
88089 lines as any form of editor command.

88090 ENVIRONMENT VARIABLES

88091 The following environment variables shall affect the execution of *ex*:

88092 *COLUMNS* Override the system-selected horizontal screen size. See XBD Chapter 8 (on page
88093 173) for valid values and results when it is unset or null.

88094 *EXINIT* Determine a list of *ex* commands that are executed on editor start-up. See the
88095 EXTENDED DESCRIPTION section for more details of the initialization phase.

88096 *HOME* Determine a pathname of a directory that shall be searched for an editor start-up
88097 file named **.exrc**; see the EXTENDED DESCRIPTION section.

88098		<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
88099			
88100			
88101		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
88102			
88103		<i>LC_COLLATE</i>	
88104			Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.
88105			
88106		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries.
88107			
88108			
88109			
88110			
88111		<i>LC_MESSAGES</i>	
88112			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
88113			
88114		<i>LINES</i>	Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See XBD Chapter 8 (on page 173) for valid values and results when it is unset or null.
88115			
88116			
88117	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
88118		<i>PATH</i>	Determine the search path for the shell command specified in the <i>ex</i> editor commands ! , shell , read , and write , and the open and visual mode command ! ; see the description of command search and execution in Section 2.9.1.1 (on page 2367).
88119			
88120			
88121		<i>SHELL</i>	Determine the preferred command line interpreter for use as the default value of the shell edit option.
88122			
88123		<i>TERM</i>	Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.
88124			
88125		ASYNCHRONOUS EVENTS	
88126			The following term is used in this and following sections to specify command and asynchronous event actions:
88127			
88128		<i>complete write</i>	
88129			A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the <i>ex</i> preserve command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.
88130			
88131			
88132			
88133			
88134			The following actions shall be taken upon receipt of signals:
88135		SIGINT	If the standard input is not a terminal device, <i>ex</i> shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.
88136			
88137			Otherwise, if executing an open or visual text input mode command, <i>ex</i> in receipt of SIGINT shall behave identically to its receipt of the <ESC> character.
88138			
88139			Otherwise:

- 88140 1. If executing an *ex* text input mode command, all input lines that have been
88141 completely entered shall be resolved into the edit buffer, and any partially
88142 entered line shall be discarded.
- 88143 2. If there is a currently executing command, it shall be aborted and a message
88144 displayed. Unless otherwise specified by the *ex* or *vi* command descriptions,
88145 it is unspecified whether any lines modified by the executing command
88146 appear modified, or as they were before being modified by the executing
88147 command, in the buffer.
- 88148 If the currently executing command was a motion command, its associated
88149 command shall be discarded.
- 88150 3. If in open or visual command mode, the terminal shall be alerted.
- 88151 4. The editor shall then return to command mode.
- 88152 SIGCONT The screen shall be refreshed if in open or visual mode.
- 88153 SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt
88154 to save the edit buffer so that it can be recovered later using the `-r` option or the *ex*
88155 **recover** command. The editor shall not write the file or return to command or text
88156 input mode, and shall terminate with a non-zero exit status.
- 88157 SIGTERM Refer to SIGHUP.
- 88158 The action taken for all other signals is unspecified.
- 88159 **STDOUT**
88160 The standard output shall be used only for writing prompts to the user, for informational
88161 messages, and for writing lines from the file.
- 88162 **STDERR**
88163 The standard error shall be used only for diagnostic messages.
- 88164 **OUTPUT FILES**
88165 The output from *ex* shall be text files.
- 88166 **EXTENDED DESCRIPTION**
88167 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing
88168 capabilities available in *ex*.
- 88169 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such
88170 as inverse video), the message shall be written in standout mode. If the terminal does not
88171 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the
88172 error message.
- 88173 By default, *ex* shall start in command mode, which shall be indicated by a `:` prompt; see the
88174 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;
88175 it can be exited (and command mode re-entered) by typing a `<period>` (`'.'`) alone at the
88176 beginning of a line.

88177 **Initialization in ex and vi**

88178 The following symbols are used in this and following sections to specify locations in the edit
88179 buffer:

88180 *alternate and current pathnames*

88181 Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*
88182 commands that take filenames as arguments shall set them as follows:

- 88183 1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**
88184 command replaces the contents of the edit buffer.
 - 88185 a. If the command replaces the contents of the edit buffer, the current pathname
88186 shall be set to the *file* argument or the file indicated by the tag, and the
88187 alternate pathname shall be set to the previous value of the current pathname.
 - 88188 b. Otherwise, the alternate pathname shall be set to the *file* argument.
- 88189 2. If a *file* argument is specified to the *ex* **next** command:
 - 88190 a. If the command replaces the contents of the edit buffer, the current pathname
88191 shall be set to the first *file* argument, and the alternate pathname shall be set to
88192 the previous value of the current pathname.
- 88193 3. If a *file* argument is specified to the *ex* **file** command, the current pathname shall be
88194 set to the *file* argument, and the alternate pathname shall be set to the previous value
88195 of the current pathname.
- 88196 4. If a *file* argument is specified to the *ex* **read** and **write** commands (that is, when
88197 reading or writing a file, and not to the program named by the **shell** edit option), or a
88198 *file* argument is specified to the *ex* **xit** command:
 - 88199 a. If the current pathname has no value, the current pathname shall be set to the
88200 *file* argument.
 - 88201 b. Otherwise, the alternate pathname shall be set to the *file* argument.

88202 If the alternate pathname is set to the previous value of the current pathname when the
88203 current pathname had no previous value, then the alternate pathname shall have no value
88204 as a result.

88205 *current line*

88206 The line of the edit buffer referenced by the cursor. Each command description specifies the
88207 current line after the command has been executed, as the *current line value*. When the edit
88208 buffer contains no lines, the current line shall be zero; see [Addressing in ex](#) (on page 2703).

88209 *current column*

88210 The current display line column occupied by the cursor. (The columns shall be numbered
88211 beginning at 1.) Each command description specifies the current column after the command
88212 has been executed, as the *current column value*. This column is an *ideal* column that is
88213 remembered over the lifetime of the editor. The actual display line column upon which the
88214 cursor rests may be different from the current column; see the cursor positioning discussion
88215 in [Command Descriptions in vi](#) (on page 3376).

88216 *set to non-<blank>*

88217 A description for a current column value, meaning that the current column shall be set to
88218 the last display line column on which is displayed any part of the first non-<blank> of the
88219 line. If the line has no non-<blank> non-<newline> characters, the current column shall be
88220 set to the last display line column on which is displayed any part of the last non-<newline>

88221 character in the line. If the line is empty, the current column shall be set to column position
88222 1.

88223 The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual
88224 mode, the length of lines in the edit buffer may be limited to the number of characters that will
88225 fit in the display. If either limit is exceeded during editing, an error message shall be written. If
88226 either limit is exceeded by a line read in from a file, an error message shall be written and the
88227 edit session may be terminated.

88228 If the editor stops running due to any reason other than a user command, and the edit buffer has
88229 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous
88230 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

88231 During initialization (before the first file is copied into the edit buffer or any user commands
88232 from the terminal are processed) the following shall occur:

- 88233 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands
88234 contained in that variable.
- 88235 2. If the *EXINIT* variable is not set, and all of the following are true:
 - 88236 a. The *HOME* environment variable is not null and not empty.
 - 88237 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:
 - 88238 i. Exists
 - 88239 ii. Is owned by the same user ID as the real user ID of the process or the
88240 process has appropriate privileges
 - 88241 iii. Is not writable by anyone other than the owner

88242 the editor shall execute the *ex* commands contained in that file.

- 88243 3. If and only if all of the following are true:
 - 88244 a. The current directory is not referred to by the *HOME* environment variable.
 - 88245 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in
88246 the directory referred to by the *HOME* environment variable sets the editor option
88247 **exrc**.
 - 88248 c. The *.exrc* file in the current directory:
 - 88249 i. Exists
 - 88250 ii. Is owned by the same user ID as the real user ID of the process, or by one of
88251 a set of implementation-defined user IDs
 - 88252 iii. Is not writable by anyone other than the owner

88253 the editor shall attempt to execute the *ex* commands contained in that file.

88254 Lines in any *.exrc* file that are blank lines shall be ignored. If any *.exrc* file exists, but is not read
88255 for ownership or permission reasons, it shall be an error.

88256 After the *EXINIT* variable and any *.exrc* files are processed, the first file specified by the user
88257 shall be edited, as follows:

- 88258 1. If the user specified the **-t** option, the effect shall be as if the *ex tag* command was entered
88259 with the specified argument, with the exception that if tag processing does not result in a
88260 file to edit, the effect shall be as described in step 3. below.

- 88261 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if
88262 the *ex edit* command was entered with the first of those arguments as its *file* argument.
- 88263 3. Otherwise, the effect shall be as if the *ex edit* command was entered with a nonexistent
88264 filename as its *file* argument. It is unspecified whether this action shall set the current
88265 pathname. In an implementation where this action does not set the current pathname, any
88266 editor command using the current pathname shall fail until an editor command sets the
88267 current pathname.

88268 If the *-r* option was specified, the first time a file in the initial argument list or a file specified by
88269 the *-t* option is edited, if recovery information has previously been saved about it, that
88270 information shall be recovered and the editor shall behave as if the contents of the edit buffer
88271 have already been modified. If there are multiple instances of the file to be recovered, the one
88272 most recently saved shall be recovered, and an informational message that there are previous
88273 versions of the file that can be recovered shall be written. If no recovery information about a file
88274 is available, an informational message to this effect shall be written, and the edit shall proceed as
88275 usual.

88276 If the *-c* option was specified, the first time a file that already exists (including a file that might
88277 not exist but for which recovery information is available, when the *-r* option is specified)
88278 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of
88279 the edit buffer, the current column shall be set to non-<blank>, and the *ex* commands specified
88280 with the *-c* option shall be executed. In this case, the current line and current column shall not
88281 be set as described for the command associated with the replacement or initialization of the edit
88282 buffer contents. However, if the *-t* option or a **tag** command is associated with this action, the *-c*
88283 option commands shall be executed and then the movement to the tag shall be performed.

88284 The current argument list shall initially be set to the filenames specified by the user on the
88285 command line. If no filenames are specified by the user, the current argument list shall be empty.
88286 If the *-t* option was specified, it is unspecified whether any filename resulting from tag
88287 processing shall be prepended to the current argument list. In the case where the filename is
88288 added as a prefix to the current argument list, the current argument list reference shall be set to
88289 that filename. In the case where the filename is not added as a prefix to the current argument
88290 list, the current argument list reference shall logically be located before the first of the filenames
88291 specified on the command line (for example, a subsequent *ex next* command shall edit the first
88292 filename from the command line). If the *-t* option was not specified, the current argument list
88293 reference shall be to the first of the filenames on the command line.

88294 Addressing in *ex*

88295 Addressing in *ex* relates to the current line and the current column; the address of a line is its
88296 1-based line number, the address of a column is its 1-based count from the beginning of the line.
88297 Generally, the current line is the last line affected by a command. The current line number is the
88298 address of the current line. In each command description, the effect of the command on the
88299 current line number and the current column is described.

88300 Addresses are constructed as follows:

- 88301 1. The character ' .' (period) shall address the current line.
- 88302 2. The character ' \$ ' shall address the last line of the edit buffer.
- 88303 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 88304 4. The address "'x" refers to the line marked with the mark name character 'x', which
88305 shall be a lowercase letter from the portable character set, the backquote character, or the
88306 single-quote character. It shall be an error if the line that was marked is not currently

88307 present in the edit buffer or the mark has not been set. Lines can be marked with the *ex*
88308 **mark** or **k** commands, or the *vi* **m** command.

88309 5. A regular expression enclosed by <slash> characters ('/') shall address the first line
88310 found by searching forwards from the line following the current line toward the end of
88311 the edit buffer and stopping at the first line for which the line excluding the terminating
88312 <newline> matches the regular expression. As stated in [Regular Expressions in ex](#) (on
88313 page 2734), an address consisting of a null regular expression delimited by <slash>
88314 characters ("/") shall address the next line for which the line excluding the terminating
88315 <newline> matches the last regular expression encountered. In addition, the second
88316 <slash> can be omitted at the end of a command line. If the **wrapsan** edit option is set,
88317 the search shall wrap around to the beginning of the edit buffer and continue up to and
88318 including the current line, so that the entire edit buffer is searched. Within the regular
88319 expression, the sequence "\/" shall represent a literal <slash> instead of the regular
88320 expression delimiter.

88321 6. A regular expression enclosed in <question-mark> characters ('?') shall address the first
88322 line found by searching backwards from the line preceding the current line toward the
88323 beginning of the edit buffer and stopping at the first line for which the line excluding the
88324 terminating <newline> matches the regular expression. An address consisting of a null
88325 regular expression delimited by <question-mark> characters ("??") shall address the
88326 previous line for which the line excluding the terminating <newline> matches the last
88327 regular expression encountered. In addition, the second <question-mark> can be omitted
88328 at the end of a command line. If the **wrapsan** edit option is set, the search shall wrap
88329 around from the beginning of the edit buffer to the end of the edit buffer and continue up
88330 to and including the current line, so that the entire edit buffer is searched. Within the
88331 regular expression, the sequence "\?" shall represent a literal <question-mark> instead
88332 of the RE delimiter.

88333 7. A <plus-sign> ('+') or a <hyphen-minus> ('-') followed by a decimal number shall
88334 address the current line plus or minus the number. A '+' or '-' not followed by a
88335 decimal number shall address the current line plus or minus 1.

88336 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.
88337 Address offsets are constructed as follows:

88338 1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the
88339 indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal
88340 number shall add (subtract) 1 to (from) the address.

88341 2. A decimal number shall add the indicated number of lines to the address.

88342 It shall not be an error for an intermediate address value to be less than zero or greater than the
88343 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
88344 greater than the last line in the edit buffer.

88345 Commands take zero, one, or two addresses; see the descriptions of *laddr* and *2addr* in
88346 [Command Descriptions in ex](#) (on page 2710). If more than the required number of addresses are
88347 provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than
88348 the required number of addresses are provided to a command, the addresses specified first shall
88349 be evaluated and then discarded until the maximum number of valid addresses remain.

88350 Addresses shall be separated from each other by a <comma> (',') or a <semicolon> (';'). If
88351 no address is specified before or after a <comma> or <semicolon> separator, it shall be as if the
88352 address of the current line was specified before or after the separator. In the case of a
88353 <semicolon> separator, the current line ('.') shall be set to the first address, and only then will

88354 the next address be calculated. This feature can be used to determine the starting line for
88355 forwards and backwards searches (see rules 5. and 6.).

88356 A <percent-sign> (' % ') shall be equivalent to entering the two addresses " 1 , \$ " .

88357 Any delimiting <blank> characters between addresses, address separators, or address offsets
88358 shall be discarded.

88359 **Command Line Parsing in ex**

88360 The following symbol is used in this and following sections to describe parsing behavior:

88361 *escape* If a character is referred to as ``<backslash>-escaped'' or ``<control>-V-escaped'', it
88362 shall mean that the character acquired or lost a special meaning by virtue of being
88363 preceded, respectively, by a <backslash> or <control>-V character. Unless
88364 otherwise specified, the escaping character shall be discarded at that time and shall
88365 not be further considered for any purpose.

88366 Command-line parsing shall be done in the following steps. For each step, characters already
88367 evaluated shall be ignored; that is, the phrase ``leading character'' refers to the next character
88368 that has not yet been evaluated.

- 88369 1. Leading <colon> characters shall be skipped.
- 88370 2. Leading <blank> characters shall be skipped.
- 88371 3. If the leading character is a double-quote character, the characters up to and including the
88372 next non-<backslash>-escaped <newline> shall be discarded, and any subsequent
88373 characters shall be parsed as a separate command.
- 88374 4. Leading characters that can be interpreted as addresses shall be evaluated; see
88375 [Addressing in ex](#) (on page 2703).
- 88376 5. Leading <blank> characters shall be skipped.
- 88377 6. If the next character is a <vertical-line> character or a <newline>:
 - 88378 a. If the next character is a <newline>:
 - 88379 i. If *ex* is in open or visual mode, the current line shall be set to the last
88380 address specified, if any.
 - 88381 ii. Otherwise, if the last command was terminated by a <vertical-line>
88382 character, no action shall be taken; for example, the command
88383 " | | <newline> " shall execute two implied commands, not three.
 - 88384 iii. Otherwise, step 6.b. shall apply.
 - 88385 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **l**
88386 flags specified to any *ex* command shall be remembered and shall apply to this
88387 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the
88388 remembered flags to #, nothing, and **l**, respectively, plus any other flags specified
88389 for that execution of the **number**, **print**, or **list** command.

88390 If *ex* is not currently performing a **global** or **v** command, and no address or count
88391 is specified, the current line shall be incremented by 1 before the command is
88392 executed. If incrementing the current line would result in an address past the last
88393 line in the edit buffer, the command shall fail, and the increment shall not happen.

- 88394 c. The <newline> or <vertical-line> character shall be discarded and any subsequent
88395 characters shall be parsed as a separate command.
- 88396 7. The command name shall be comprised of the next character (if the character is not
88397 alphabetic), or the next character and any subsequent alphabetic characters (if the
88398 character is alphabetic), with the following exceptions:
- 88399 a. Commands that consist of any prefix of the characters in the command name
88400 **delete**, followed immediately by any of the characters 'l', 'p', '+', '-', or '#'
88401 shall be interpreted as a **delete** command, followed by a <blank>, followed by the
88402 characters that were not part of the prefix of the **delete** command. The maximum
88403 number of characters shall be matched to the command name **delete**; for example,
88404 "del" shall not be treated as "de" followed by the flag l.
- 88405 b. Commands that consist of the character 'k', followed by a character that can be
88406 used as the name of a mark, shall be equivalent to the mark command followed by
88407 a <blank>, followed by the character that followed the 'k'.
- 88408 c. Commands that consist of the character 's', followed by characters that could be
88409 interpreted as valid options to the **s** command, shall be the equivalent of the **s**
88410 command, without any pattern or replacement values, followed by a <blank>,
88411 followed by the characters after the 's'.
- 88412 8. The command name shall be matched against the possible command names, and a
88413 command name that contains a prefix matching the characters specified by the user shall
88414 be the executed command. In the case of commands where the characters specified by the
88415 user could be ambiguous, the executed command shall be as follows:

88416
88417
88418
88419
88420
88421

a	append	n	next	t	t
c	change	p	print	u	undo
ch	change	pr	print	un	undo
e	edit	r	read	v	v
m	move	re	read	w	write
ma	mark	s	s		

88422
88423
88424

Implementation extensions with names causing similar ambiguities shall not be checked for a match until all possible matches for commands specified by POSIX.1-2017 have been checked.

88425
88426
88427
88428
88429

9. If the command is a **!** command, or if the command is a **read** command followed by zero or more <blank> characters and a **!**, or if the command is a **write** command followed by one or more <blank> characters and a **!**, the rest of the command shall include all characters up to a non-<backslash>-escaped <newline>. The <newline> shall be discarded and any subsequent characters shall be parsed as a separate *ex* command.

88430
88431

10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in open or visual mode, the next part of the command shall be parsed as follows:

88432
88433

- a. Any **!** character immediately following the command shall be skipped and be part of the command.

88434

- b. Any leading <blank> characters shall be skipped and be part of the command.

88435
88436
88437

- c. If the next character is a **+**, characters up to the first non-<backslash>-escaped <newline> or non-<backslash>-escaped <blank> shall be skipped and be part of the command.

- 88438 d. The rest of the command shall be determined by the steps specified in paragraph
88439 12.
- 88440 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the
88441 command shall be parsed as follows:
- 88442 a. Any leading <blank> characters shall be skipped and be part of the command.
- 88443 b. If the next character is not an alphanumeric, double-quote, <newline>,
88444 <backslash>, or <vertical-line> character:
- 88445 i. The next character shall be used as a command delimiter.
- 88446 ii. If the command is a **global**, **open**, or **v** command, characters up to the first
88447 non-<backslash>-escaped <newline>, or first non-<backslash>-escaped
88448 delimiter character, shall be skipped and be part of the command.
- 88449 iii. If the command is an **s** command, characters up to the first
88450 non-<backslash>-escaped <newline>, or second non-<backslash>-escaped
88451 delimiter character, shall be skipped and be part of the command.
- 88452 c. If the command is a **global** or **v** command, characters up to the first
88453 non-<backslash>-escaped <newline> shall be skipped and be part of the
88454 command.
- 88455 d. Otherwise, the rest of the command shall be determined by the steps specified in
88456 paragraph 12.
- 88457 12. Otherwise:
- 88458 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,
88459 characters up to the first non-<control>-V-escaped <newline>, <vertical-line>, or
88460 double-quote character shall be skipped and be part of the command.
- 88461 b. Otherwise, characters up to the first non-<backslash>-escaped <newline>,
88462 <vertical-line>, or double-quote character shall be skipped and be part of the
88463 command.
- 88464 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
88465 ended at a <vertical-line> character, any subsequent characters, up to the next
88466 non-<backslash>-escaped <newline> shall be used as input text to the command.
- 88467 d. If the command was ended by a double-quote character, all subsequent characters,
88468 up to the next non-<backslash>-escaped <newline>, shall be discarded.
- 88469 e. The terminating <newline> or <vertical-line> character shall be discarded and any
88470 subsequent characters shall be parsed as a separate *ex* command.

88471 Command arguments shall be parsed as described by the Synopsis and Description of each
88472 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,
88473 which must follow the command name without intervening <blank> characters, and where it
88474 would otherwise be ambiguous. For example, *count* and *flag* arguments need not be
88475 <blank>-separated because "d22p" is not ambiguous, but *file* arguments to the *ex next*
88476 command must be separated by one or more <blank> characters. Any <blank> in command
88477 arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands can be
88478 <control>-V-escaped, in which case the <blank> shall not be used as an argument delimiter. Any
88479 <blank> in the command argument for any other command can be <backslash>-escaped, in
88480 which case that <blank> shall not be used as an argument delimiter.

88481 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,

88482 any character can be <control>-V-escaped. All such escaped characters shall be treated literally
 88483 and shall have no special meaning. Within command arguments for all other *ex* commands that
 88484 are not regular expressions or replacement strings, any character that would otherwise have a
 88485 special meaning can be <backslash>-escaped. Escaped characters shall be treated literally,
 88486 without special meaning as shell expansion characters or '!', '%', and '#' expansion
 88487 characters. See [Regular Expressions in ex](#) (on page 2734) and [Replacement Strings in ex](#) (on page
 88488 2735) for descriptions of command arguments that are regular expressions or replacement
 88489 strings.

88490 Non-<backslash>-escaped '%' characters appearing in *file* arguments to any *ex* command shall
 88491 be replaced by the current pathname; unescaped '#' characters shall be replaced by the
 88492 alternate pathname. It shall be an error if '%' or '#' characters appear unescaped in an
 88493 argument and their corresponding values are not set.

88494 Non-<backslash>-escaped '!' characters in the arguments to either the *ex !* command or the
 88495 open and visual mode *!* command, or in the arguments to the *ex read* command, where the first
 88496 non-<blank> after the command name is a '!' character, or in the arguments to the *ex write*
 88497 command where the command name is followed by one or more <blank> characters and the
 88498 first non-<blank> after the command name is a '!' character, shall be replaced with the
 88499 arguments to the last of those three commands as they appeared after all unescaped '%', '#',
 88500 and '!' characters were replaced. It shall be an error if '!' characters appear unescaped in one
 88501 of these commands and there has been no previous execution of one of these commands.

88502 If an error occurs during the parsing or execution of an *ex* command:

88503 An informational message to this effect shall be written. Execution of the *ex* command shall
 88504 stop, and the cursor (for example, the current line and column) shall not be further
 88505 modified.

88506 If the *ex* command resulted from a map expansion, all characters from that map expansion
 88507 shall be discarded, except as otherwise specified by the **map** command.

88508 Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment
 88509 variable, a **.exrc** file, a **:source** command, a **-c** option, or a **+command** specified to an *ex edit*,
 88510 **ex next**, or **visual** command, no further commands from the source of the commands shall
 88511 be executed.

88512 Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v**
 88513 command, no further commands caused by the execution of the buffer or the **global** or **v**
 88514 command shall be executed.

88515 Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and
 88516 including the next non-<backslash>-escaped <newline> shall be discarded.

88517 **Input Editing in ex**

88518 The following symbol is used in this and the following sections to specify command actions:

88519 *word* In the POSIX locale, a word consists of a maximal sequence of letters, digits, and
 88520 underscores, delimited at both ends by characters other than letters, digits, or
 88521 underscores, or by the beginning or end of a line or the edit buffer.

88522 When accepting input characters from the user, in either *ex* command mode or *ex* text input
 88523 mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces
 88524 volume of POSIX.1-2017.

88525 If in *ex* text input mode:

- 88526 1. If the **number** edit option is set, *ex* shall prompt for input using the line number that
88527 would be assigned to the line if it is entered, in the format specified for the *ex* **number**
88528 command.
- 88529 2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters,
88530 as described by the **autoindent** edit option. **autoindent** characters shall follow the line
88531 number, if any.

88532 If in *ex* command mode:

- 88533 1. If the **prompt** edit option is set, input shall be prompted for using a single ' : ' character;
88534 otherwise, there shall be no prompt.

88535 The input characters in the following sections shall have the following effects on the input line.

88536 **Scroll**

88537 *Synopsis:* eof

88538 See the description of the *stty* eof character in *stty*.

88539 If in *ex* command mode:

88540 If the eof character is the first character entered on the line, the line shall be evaluated as if
88541 it contained two characters: a <control>-D and a <newline>.

88542 Otherwise, the eof character shall have no special meaning.

88543 If in *ex* text input mode:

88544 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be
88545 modified so that a part of the next text input character will be displayed on the first
88546 column in the line after the previous **shiftwidth** edit option column boundary, and the
88547 user shall be prompted again for input for the same line.

88548 Otherwise, if the cursor follows a ' 0 ', which follows an **autoindent** character, and the ' 0 '
88549 was the previous text input character, the ' 0 ' and all **autoindent** characters in the line
88550 shall be discarded, and the user shall be prompted again for input for the same line.

88551 Otherwise, if the cursor follows a ' ^ ', which follows an **autoindent** character, and the ' ^ '
88552 was the previous text input character, the ' ^ ' and all **autoindent** characters in the line
88553 shall be discarded, and the user shall be prompted again for input for the same line. In
88554 addition, the **autoindent** level for the next input line shall be derived from the same line
88555 from which the **autoindent** level for the current input line was derived.

88556 Otherwise, if there are no **autoindent** or text input characters in the line, the eof character
88557 shall be discarded.

88558 Otherwise, the eof character shall have no special meaning.

88559 **<newline>**

88560 *Synopsis:* <newline>
88561 <control>-J

88562 If in *ex* command mode:

88563 Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for
88564 this purpose.

88565 If in *ex* text input mode:

88566 Terminate the current line. If there are no characters other than **autoindent** characters on
88567 the line, all characters on the line shall be discarded.

88568 Prompt for text input on a new line after the current line. If the **autoindent** edit option is
88569 set, an appropriate number of **autoindent** characters shall be added as a prefix to the line
88570 as described by the *ex* **autoindent** edit option.

88571 **<backslash>**

88572 *Synopsis:* <backslash>

88573 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any
88574 special meaning that it may have to the editor during text input mode. The <backslash>
88575 character shall be retained and evaluated when the command line is parsed, or retained and
88576 included when the input text becomes part of the edit buffer.

88577 **<control>-V**

88578 *Synopsis:* <control>-V

88579 Allow the entry of any subsequent character as a literal character, removing any special meaning
88580 that it may have to the editor during text input mode. The <control>-V character shall be
88581 discarded before the command line is parsed or the input text becomes part of the edit buffer.

88582 If the “literal next” functionality is performed by the underlying system, it is implementation-
88583 defined whether a character other than <control>-V performs this function.

88584 **<control>-W**

88585 *Synopsis:* <control>-W

88586 Discard the <control>-W, and the word previous to it in the input line, including any <blank>
88587 characters following the word and preceding the <control>-W. If the “word erase” functionality
88588 is performed by the underlying system, it is implementation-defined whether a character other
88589 than <control>-W performs this function.

88590 **Command Descriptions in ex**

88591 The following symbols are used in this section to represent command modifiers. Some of these
88592 modifiers can be omitted, in which case the specified defaults shall be used.

88593 *laddr* A single line address, given in any of the forms described in [Addressing in ex](#) (on
88594 page 2703); the default shall be the current line (' . '), unless otherwise specified.

88595 If the line address is zero, it shall be an error, unless otherwise specified in the
88596 following command descriptions.

88597		If the edit buffer is empty, and the address is specified with a command other than
88598		= , append , insert , open , put , read , or visual , or the address is not zero, it shall be
88599		an error.
88600	<i>2addr</i>	Two addresses specifying an inclusive range of lines. If no addresses are specified,
88601		the default for <i>2addr</i> shall be the current line only (" . , . "), unless otherwise
88602		specified in the following command descriptions. If one address is specified, <i>2addr</i>
88603		shall specify that line only, unless otherwise specified in the following command
88604		descriptions.
88605		It shall be an error if the first address is greater than the second address.
88606		If the edit buffer is empty, and the two addresses are specified with a command
88607		other than the ! , write , wq , or xit commands, or either address is not zero, it shall
88608		be an error.
88609	<i>count</i>	A positive decimal number. If <i>count</i> is specified, it shall be equivalent to specifying
88610		an additional address to the command, unless otherwise specified by the following
88611		command descriptions. The additional address shall be equal to the last address
88612		specified to the command (either explicitly or by default) plus <i>count</i> –1.
88613		If this would result in an address greater than the last line of the edit buffer, it shall
88614		be corrected to equal the last line of the edit buffer.
88615	<i>flags</i>	One or more of the characters '+', '-', '#', 'p', or 'l' (ell). The flag characters
88616		can be <blank>-separated, and in any order or combination. The characters '#',
88617		'p', and 'l' shall cause lines to be written in the format specified by the print
88618		command with the specified <i>flags</i> .
88619		The lines to be written are as follows:
88620		1. All edit buffer lines written during the execution of the <i>ex</i> & , ~ , list , number ,
88621		open , print , s , visual , and z commands shall be written as specified by <i>flags</i> .
88622		2. After the completion of an <i>ex</i> command with a flag as an argument, the
88623		current line shall be written as specified by <i>flags</i> , unless the current line was
88624		the last line written by the command.
88625		The characters '+' and '-' cause the value of the current line after the execution
88626		of the <i>ex</i> command to be adjusted by the offset address as described in Addressing
88627		in ex (on page 2703). This adjustment shall occur before the current line is written
88628		as described in 2. above.
88629		The default for <i>flags</i> shall be none.
88630	<i>buffer</i>	One of a number of named areas for holding text. The named buffers are specified
88631		by the alphanumeric characters of the POSIX locale. There shall also be one
88632		"unnamed" buffer. When no buffer is specified for editor commands that use a
88633		buffer, the unnamed buffer shall be used. Commands that store text into buffers
88634		shall store the text as it was before the command took effect, and shall store text
88635		occurring earlier in the file before text occurring later in the file, regardless of how
88636		the text region was specified. Commands that store text into buffers shall store the
88637		text into the unnamed buffer as well as any specified buffer.
88638		In <i>ex</i> commands, buffer names are specified as the name by itself. In open or visual
88639		mode commands the name is preceded by a double-quote ('"') character.
88640		If the specified buffer name is an uppercase character, and the buffer contents are
88641		to be modified, the buffer shall be appended to rather than being overwritten. If

88642 the buffer is not being modified, specifying the buffer name in lowercase and
88643 uppercase shall have identical results.

88644 There shall also be buffers named by the numbers 1 through 9. In open and visual
88645 mode, if a region of text including characters from more than a single line is being
88646 modified by the *vi* **c** or **d** commands, the motion character associated with the **c** or
88647 **d** commands specifies that the buffer text shall be in line mode, or the commands
88648 **%**, **`**, **/**, **?**, **(**, **)**, **N**, **n**, **{**, or **}** are used to define a region of text for the **c** or **d** commands,
88649 the contents of buffers 1 through 8 shall be moved into the buffer named by the
88650 next numerically greater value, the contents of buffer 9 shall be discarded, and the
88651 region of text shall be copied into buffer 1. This shall be in addition to copying the
88652 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can
88653 be specified as a source buffer for open and visual mode commands; however,
88654 specifying a numeric buffer as the write target of an open or visual mode
88655 command shall have unspecified results.

88656 The text of each buffer shall have the characteristic of being in either line or
88657 character mode. Appending text to a non-empty buffer shall set the mode to match
88658 the characteristic of the text being appended. Appending text to a buffer shall
88659 cause the creation of at least one additional line in the buffer. All text stored into
88660 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers
88661 as the source of text specify individually how buffers of different modes are
88662 handled. Each open or visual mode command that uses buffers for any purpose
88663 specifies individually the mode of the text stored into the buffer and how buffers
88664 of different modes are handled.

88665 *file* Command text used to derive a pathname. The default shall be the current
88666 pathname, as defined previously, in which case, if no current pathname has yet
88667 been established it shall be an error, except where specifically noted in the
88668 individual command descriptions that follow. If the command text contains any of
88669 the characters **'**, **~**, **{**, **[**, *****, **?**, **\$**, **"**, backquote, single-quote, and
88670 **<backslash>**, it shall be subjected to the process of "shell expansions", as described
88671 below; if more than a single pathname results and the command expects only one,
88672 it shall be an error.

88673 The process of shell expansions in the editor shall be done as follows. The *ex* utility
88674 shall pass two arguments to the program named by the shell edit option; the first
88675 shall be **-c**, and the second shall be the string "echo" and the command text as a
88676 single argument. The standard output and standard error of that command shall
88677 replace the command text.

88678 **!** A character that can be appended to the command name to modify its operation,
88679 as detailed in the individual command descriptions. With the exception of the *ex*
88680 **read**, **write**, and **!** commands, the **!** character shall only act as a modifier if there
88681 are no **<blank>** characters between it and the command name.

88682 *remembered search direction*

88683 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in
88684 the edit buffer based on a remembered search direction, which is initially unset,
88685 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* **/** and **?** commands.

88686 **Abbreviate**88687 *Synopsis:* `ab[breviate] [lhs rhs]`88688 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.88689 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
88690 characters and <blank> characters shall not be restricted. Additional restrictions shall be
88691 implementation-defined.88692 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character
88693 shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.88694 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a
88695 <control>-V character is entered after a word character, a check shall be made for a set of
88696 characters matching *lhs*, in the text input entered during this command. If it is found, the effect
88697 shall be as if *rhs* was entered instead of *lhs*.

88698 The set of characters that are checked is defined as follows:

- 88699 1. If there are no characters inserted before the word and non-word or <ESC> characters
88700 that triggered the check, the set of characters shall consist of the word character.
- 88701 2. If the character inserted before the word and non-word or <ESC> characters that
88702 triggered the check is a word character, the set of characters shall consist of the characters
88703 inserted immediately before the triggering characters that are word characters, plus the
88704 triggering word character.
- 88705 3. If the character inserted before the word and non-word or <ESC> characters that
88706 triggered the check is not a word character, the set of characters shall consist of the
88707 characters that were inserted before the triggering characters that are neither <blank>
88708 characters nor word characters, plus the triggering word character.

88709 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**
88710 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the
88711 effect of the command shall be as if the replacement had not occurred.88712 *Current line:* Unchanged.88713 *Current column:* Unchanged.88714 **Append**88715 *Synopsis:* `[laddr] a[ppend] [!]`88716 Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is
88717 specified, the text shall be placed at the beginning of the edit buffer.88718 This command shall be affected by the **number** and **autoindent** edit options; following the
88719 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
88720 duration of this command only.88721 *Current line:* Set to the last input line; if no lines were input, set to the specified line, or to the first
88722 line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.88723 *Current column:* Set to non-<blank>.

88724 **Arguments**88725 *Synopsis:* ar[gs]88726 Write the current argument list, with the current argument-list entry, if any, between '[' and
88727 ']' characters.88728 *Current line:* Unchanged.88729 *Current column:* Unchanged.88730 **Change**88731 *Synopsis:* [2addr] c[hange][!][count]88732 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall
88733 be copied into the unnamed buffer, which shall become a line mode buffer.88734 This command shall be affected by the **number** and **autoindent** edit options; following the
88735 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
88736 duration of this command only.88737 *Current line:* Set to the last input line; if no lines were input, set to the line before the first
88738 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to
88739 zero if the edit buffer is empty.88740 *Current column:* Set to non-<blank>.88741 **Change Directory**88742 *Synopsis:* chd[ir][!][directory]

88743 cd[!][directory]

88744 Change the current working directory to *directory*.88745 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null
88746 and non-empty value, *directory* shall default to the value named in the *HOME* environment
88747 variable. If the *HOME* environment variable is empty or is undefined, the default value of
88748 *directory* is implementation-defined.88749 If no '!' is appended to the command name, and the edit buffer has been modified since the
88750 last complete write, and the current pathname does not begin with a '/', it shall be an error.88751 *Current line:* Unchanged.88752 *Current column:* Unchanged.88753 **Copy**88754 *Synopsis:* [2addr] co[py] laddr [flags]

88755 [2addr] t laddr [flags]

88756 Copy the specified lines after the specified destination line; line zero specifies that the lines shall
88757 be placed at the beginning of the edit buffer.88758 *Current line:* Set to the last line copied.88759 *Current column:* Set to non-<blank>.

88760 **Delete**88761 *Synopsis:* `[2addr] d[delete] [buffer] [count] [flags]`88762 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a
88763 line-mode buffer.88764 Flags can immediately follow the command name; see [Command Line Parsing in ex](#) (on page
88765 2705).88766 *Current line:* Set to the line following the deleted lines, or to the last line in the edit buffer if that
88767 line is past the end of the edit buffer, or to zero if the edit buffer is empty.88768 *Current column:* Set to non-<blank>.88769 **Edit**88770 *Synopsis:* `e[dit] [!] [+command] [file]`
88771 `ex [!] [+command] [file]`88772 If no '!' is appended to the command name, and the edit buffer has been modified since the
88773 last complete write, it shall be an error.88774 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,
88775 and set the current pathname to *file*. If *file* is not specified, replace the current contents of the
88776 edit buffer with the current contents of the file named by the current pathname. If for any reason
88777 the current contents of the file cannot be accessed, the edit buffer shall be empty.88778 The *+command* option shall be <blank>-delimited; <blank> characters within the *+command* can
88779 be escaped by preceding them with a <backslash> character. The *+command* shall be interpreted
88780 as an *ex* command immediately after the contents of the edit buffer have been replaced and the
88781 current line and column have been set.

88782 If the edit buffer is empty:

88783 *Current line:* Set to 0.88784 *Current column:* Set to 1.88785 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:88786 *Current line:* Set to the last line of the edit buffer.88787 *Current column:* Set to non-<blank>.88788 Otherwise, if *file* is omitted or results in the current pathname:88789 *Current line:* Set to the first line of the edit buffer.88790 *Current column:* Set to non-<blank>.88791 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if
88792 the file was previously edited, the line and column may be set as follows:88793 *Current line:* Set to the last value held when that file was last edited. If this value is not a valid
88794 line in the new edit buffer, set to the first line of the edit buffer.88795 *Current column:* If the current line was set to the last value held when the file was last edited, set
88796 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid
88797 column in the new edit buffer, set to non-<blank>.

88798 Otherwise:
 88799 *Current line*: Set to the first line of the edit buffer.
 88800 *Current column*: Set to non-<blank>.

88801 File

88802 *Synopsis*: *f* [*file*] [*file*]

88803 If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and
 88804 the current pathname shall be set to *file*.

88805 Write an informational message. If the file has a current pathname, it shall be included in this
 88806 message; otherwise, the message shall indicate that there is no current pathname. If the edit
 88807 buffer contains lines, the current line number and the number of lines in the edit buffer shall be
 88808 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If
 88809 the edit buffer has been modified since the last complete write, this fact shall be included in this
 88810 message. If the **readonly** edit option is set, this fact shall be included in this message. The
 88811 message may contain other unspecified information.

88812 *Current line*: Unchanged.

88813 *Current column*: Unchanged.

88814 Global

88815 *Synopsis*: [*2addr*] **g**[*lobal*] /*pattern*/ [*commands*]
 88816 [*2addr*] **v** /*pattern*/ [*commands*]

88817 The optional **!** character after the **global** command shall be the same as executing the **v**
 88818 command.

88819 If *pattern* is empty (for example, *"/"*) or not specified, the last regular expression used in the
 88820 editor command shall be used as the *pattern*. The *pattern* can be delimited by <slash> characters
 88821 (shown in the Synopsis), as well as any non-alphanumeric or non-<blank> other than
 88822 <backslash>, <vertical-line>, <newline>, or double-quote.

88823 If no lines are specified, the lines shall default to the entire file.

88824 The **global** and **v** commands are logically two-pass operations. First, mark the lines within the
 88825 specified lines for which the line excluding the terminating <newline> matches (**global**) or does
 88826 not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by
 88827 *commands*, with the current line ('.') set to each marked line. If an error occurs during this
 88828 process, or the contents of the edit buffer are replaced (for example, by the *ex* **edit** command) an
 88829 error message shall be written and no more commands resulting from the execution of this
 88830 command shall be processed.

88831 Multiple *ex* commands can be specified by entering multiple commands on a single line using a
 88832 <vertical-line> to delimit them, or one per line, by escaping each <newline> with a <backslash>.

88833 If no commands are specified:

- 88834 1. If in *ex* command mode, it shall be as if the **print** command were specified.
- 88835 2. Otherwise, no command shall be executed.

88836 For the **append**, **change**, and **insert** commands, the input text shall be included as part of the
 88837 command, and the terminating <period> can be omitted if the command ends the list of
 88838 commands. The **open** and **visual** commands can be specified as one of the commands, in which

88839 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode
 88840 is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open
 88841 or visual mode reentered, until the list of marked lines is exhausted.

88842 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted
 88843 by commands executed for lines occurring earlier in the file than the marked lines. In this case,
 88844 no commands shall be executed for the deleted lines.

88845 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

88846 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**
 88847 command.

88848 *Current line*: If no commands executed, set to the last marked line. Otherwise, as specified for the
 88849 executed *ex* commands.

88850 *Current column*: If no commands are executed, set to non-<blank>; otherwise, as specified for the
 88851 individual *ex* commands.

88852 **Insert**

88853 *Synopsis*: `[laddr] i[nsert][!]`

88854 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero
 88855 or 1, the text shall be placed at the beginning of the edit buffer.

88856 This command shall be affected by the **number** and **autoindent** edit options; following the
 88857 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
 88858 duration of this command only.

88859 *Current line*: Set to the last input line; if no lines were input, set to the line before the specified
 88860 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero
 88861 if the edit buffer is empty.

88862 *Current column*: Set to non-<blank>.

88863 **Join**

88864 *Synopsis*: `[2addr] j[oin][!][count][flags]`

88865 If *count* is specified:

88866 If no address was specified, the **join** command shall behave as if *2addr* were the current
 88867 line and the current line plus *count* (*.*, *.* + *count*).

88868 If one address was specified, the **join** command shall behave as if *2addr* were the specified
 88869 address and the specified address plus *count* (*addr*, *addr* + *count*).

88870 If two addresses were specified, the **join** command shall behave as if an additional
 88871 address, equal to the last address plus *count* -1 (*addr1*, *addr2*, *addr2* + *count* -1), was
 88872 specified.

88873 If this would result in a second address greater than the last line of the edit buffer, it shall
 88874 be corrected to be equal to the last line of the edit buffer.

88875 If no *count* is specified:

88876 If no address was specified, the **join** command shall behave as if *2addr* were the current
88877 line and the next line (*.*, *.* +1).

88878 If one address was specified, the **join** command shall behave as if *2addr* were the specified
88879 address and the next line (*addr*, *addr* +1).

88880 Join the text from the specified lines together into a single line, which shall replace the specified
88881 lines.

88882 If a '!' character is appended to the command name, the **join** shall be without modification of
88883 any line, independent of the current locale.

88884 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for
88885 each subsequent line, proceed as follows:

- 88886 1. Discard leading <space> characters from the line to be joined.
- 88887 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 88888 3. If the current line ends in a <blank>, or the first character of the line to be joined is a ') '
88889 character, join the lines without further modification.
- 88890 4. If the last character of the current line is a ' . ', join the lines with two <space> characters
88891 between them.
- 88892 5. Otherwise, join the lines with a single <space> between them.

88893 *Current line*: Set to the first line specified.

88894 *Current column*: Set to non-<blank>.

88895 List

88896 *Synopsis*: [*2addr*] l[*list*] [*count*] [*flags*]

88897 This command shall be equivalent to the *ex* command:

88898 [*2addr*] p[rint] [*count*] l[*flags*]

88899 See [Print](#) (on page 2722).

88900 Map

88901 *Synopsis*: map[!] [*lhs rhs*]

88902 If *lhs* and *rhs* are not specified:

- 88903 1. If '!' is specified, write the current list of text input mode maps.
- 88904 2. Otherwise, write the current list of command mode maps.
- 88905 3. Do nothing more.

88906 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
88907 characters and <blank> characters shall not be restricted. Additional restrictions shall be
88908 implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in
88909 which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V
88910 shall be discarded.

88911 If the character '!' is appended to the **map** command name, the mapping shall be effective
88912 during open or visual text input mode rather than **open** or **visual** command mode. This allows
88913 *lhs* to have two different **map** definitions at the same time: one for command mode and one for

- 88914 text input mode.
- 88915 For command mode mappings:
- 88916 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as
88917 part of the arguments to the command), the action shall be as if the corresponding *rhs* had
88918 been entered.
- 88919 If any character in the command, other than the first, is escaped using a <control>-V
88920 character, that character shall not be part of a match to an *lhs*.
- 88921 It is unspecified whether implementations shall support **map** commands where the *lhs* is
88922 more than a single character in length, where the first character of the *lhs* is printable.
- 88923 If *lhs* contains more than one character and the first character is '#', followed by a
88924 sequence of digits corresponding to a numbered function key, then when this function key
88925 is typed it shall be mapped to *rhs*. Characters other than digits following a '#' character
88926 also represent the function key named by the characters in the *lhs* following the '#' and
88927 may be mapped to *rhs*. It is unspecified how function keys are named or what function
88928 keys are supported.
- 88929 For text input mode mappings:
- 88930 When the *lhs* is entered as any part of text entered in open or visual text input modes, the
88931 action shall be as if the corresponding *rhs* had been entered.
- 88932 If any character in the input text is escaped using a <control>-V character, that character
88933 shall not be part of a match to an *lhs*.
- 88934 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is
88935 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or
88936 not the display appears as if the corresponding *rhs* text was entered, the effect of the
88937 command shall be as if the *lhs* text was entered.
- 88938 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,
88939 possibly matching characters before treating the already entered characters as not matching the
88940 *lhs*.
- 88941 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the
88942 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those
88943 characters shall not be remapped.
- 88944 On block-mode terminals, the mapping need not occur immediately (for example, it may occur
88945 after the terminal transmits a group of characters to the system), but it shall achieve the same
88946 results as if it occurred immediately.
- 88947 *Current line*: Unchanged.
- 88948 *Current column*: Unchanged.

88949 **Mark**

88950 *Synopsis:* `[laddr] ma[rk] character`
 88951 `[laddr] k character`

88952 Implementations shall support *character* values of a single lowercase letter of the POSIX locale
 88953 and the backquote and single-quote characters; support of other characters is implementation-
 88954 defined.

88955 If executing the *vi m* command, set the specified mark to the current line and 1-based numbered
 88956 character referenced by the current column, if any; otherwise, column position 1.

88957 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>
 88958 non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any;
 88959 otherwise, column position 1.

88960 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a
 88961 deleted line is restored by a subsequent **undo** command, any marks previously associated with
 88962 the line, which have not been reset, shall be restored as well. Any use of a mark not associated
 88963 with a current line in the edit buffer shall be an error.

88964 The marks ` and ' shall be set as described previously, immediately before the following events
 88965 occur in the editor:

- 88966 1. The use of '\$' as an *ex* address
- 88967 2. The use of a positive decimal number as an *ex* address
- 88968 3. The use of a search command as an *ex* address
- 88969 4. The use of a mark reference as an *ex* address
- 88970 5. The use of the following open and visual mode commands: <control>-], %, (,), [,], {, }
- 88971 6. The use of the following open and visual mode commands: ', **G**, **H**, **L**, **M**, **z** if the current
 88972 line will change as a result of the command
- 88973 7. The use of the open and visual mode commands: /, ?, **N**, `, **n** if the current line or column
 88974 will change as a result of the command
- 88975 8. The use of the *ex* mode commands: **z**, **undo**, **global**, **v**

88976 For rules 1., 2., 3., and 4., the ` and ' marks shall not be set if the *ex* command is parsed as
 88977 specified by rule 6.a. in [Command Line Parsing in ex](#) (on page 2705).

88978 For rules 5., 6., and 7., the ` and ' marks shall not be set if the commands are used as motion
 88979 commands in open and visual mode.

88980 For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ` and ' marks shall not be set if the command fails.

88981 The ` and ' marks shall be set as described previously, each time the contents of the edit buffer
 88982 are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex**
 88983 mode and the edit buffer is not empty, before any commands or movements (including
 88984 commands or movements specified by the `-c` or `-t` options or the `+command` argument) are
 88985 executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the *vi*
 88986 **m** command; otherwise, as if executing the *ex mark* command.

88987 When changing from **ex** mode to open or visual mode, if the ` and ' marks are not already set,
 88988 the ` and ' marks shall be set as described previously.

88989 *Current line:* Unchanged.

88990 *Current column*: Unchanged.

88991 **Move**

88992 *Synopsis*: `[2addr] m[ove] laddr [flags]`

88993 Move the specified lines after the specified destination line. A destination of line zero specifies
88994 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the
88995 destination line is within the range of lines to be moved.

88996 *Current line*: Set to the last of the moved lines.

88997 *Current column*: Set to non-<blank>.

88998 **Next**

88999 *Synopsis*: `n[ext] [!] [+command] [file ...]`

89000 If no '!' is appended to the command name, and the edit buffer has been modified since the
89001 last complete write, it shall be an error, unless the file is successfully written as specified by the
89002 **autowrite** option.

89003 If one or more files is specified:

- 89004 1. Set the argument list to the specified filenames.
- 89005 2. Set the current argument list reference to be the first entry in the argument list.
- 89006 3. Set the current pathname to the first filename specified.

89007 Otherwise:

- 89008 1. It shall be an error if there are no more filenames in the argument list after the filename
89009 currently referenced.
- 89010 2. Set the current pathname and the current argument list reference to the filename after the
89011 filename currently referenced in the argument list.

89012 Replace the contents of the edit buffer with the contents of the file named by the current
89013 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be
89014 empty.

89015 This command shall be affected by the **autowrite** and **writeany** edit options.

89016 The *+command* option shall be <blank>-delimited; <blank> characters can be escaped by
89017 preceding them with a <backslash> character. The *+command* shall be interpreted as an *ex*
89018 command immediately after the contents of the edit buffer have been replaced and the current
89019 line and column have been set.

89020 *Current line*: Set as described for the **edit** command.

89021 *Current column*: Set as described for the **edit** command.

89022 **Number**

89023 *Synopsis:* [2addr] nu[mber] [count] [flags]
 89024 [2addr] #[count] [flags]

89025 These commands shall be equivalent to the *ex* command:

89026 [2addr] p[rint] [count] #[flags]

89027 See [Print](#).

89028 **Open**

89029 *Synopsis:* [laddr] o[pen] /pattern/ [flags]

89030 This command need not be supported on block-mode terminals or terminals with insufficient
 89031 capabilities. If standard input, standard output, or standard error are not terminal devices, the
 89032 results are unspecified.

89033 Enter open mode.

89034 The trailing delimiter can be omitted from *pattern* at the end of the command line. If *pattern* is
 89035 empty (for example, "//") or not specified, the last regular expression used in the editor shall
 89036 be used as the pattern. The pattern can be delimited by <slash> characters (shown in the
 89037 *Synopsis*), as well as any alphanumeric, or non-<blank> other than <backslash>, <vertical-line>,
 89038 <newline>, or double-quote.

89039 *Current line:* Set to the specified line.

89040 *Current column:* Set to non-<blank>.

89041 **Preserve**

89042 *Synopsis:* pre[serve]

89043 Save the edit buffer in a form that can later be recovered by using the **-r** option or by using the
 89044 *ex recover* command. After the file has been preserved, a mail message shall be sent to the user.
 89045 This message shall be readable by invoking the *mailx* utility. The message shall contain the name
 89046 of the file, the time of preservation, and an *ex* command that could be used to recover the file.
 89047 Additional information may be included in the mail message.

89048 *Current line:* Unchanged.

89049 *Current column:* Unchanged.

89050 **Print**

89051 *Synopsis:* [2addr] p[rint] [count] [flags]

89052 Write the addressed lines. The behavior is unspecified if the number of columns on the display is
 89053 less than the number of columns required to write any single character in the lines being written.

89054 Non-printable characters, except for the <tab>, shall be written as implementation-defined
 89055 multi-character sequences.

89056 If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line
 89057 number in the following format:

89058 "%6dΔΔ", <line number>

89059 If the **l** flag is specified or the **list** edit option is set:

- 89060 1. The characters listed in XBD [Table 5-1](#) (on page 121) shall be written as the corresponding
89061 escape sequence.
- 89062 2. Non-printable characters not in XBD [Table 5-1](#) (on page 121) shall be written as one three-
89063 digit octal number (with a preceding <backslash>) for each byte in the character (most
89064 significant byte first).
- 89065 3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line
89066 shall be written with a preceding <backslash>.

89067 Long lines shall be folded; the length at which folding occurs is unspecified, but should be
89068 appropriate for the output terminal, considering the number of columns of the terminal.

89069 If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified
89070 whether a multi-column character at the folding position is separated; it shall not be discarded.

89071 *Current line*: Set to the last written line.

89072 *Current column*: Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

89073 Put

89074 *Synopsis*: `[laddr] pu[t] [buffer]`

89075 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line
89076 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a
89077 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

89078 *Current line*: Set to the last line entered into the edit buffer.

89079 *Current column*: Set to non-<blank>.

89080 Quit

89081 *Synopsis*: `q[uit] [!]`

89082 If no '!' is appended to the command name:

- 89083 1. If the edit buffer has been modified since the last complete write, it shall be an error.
- 89084 2. If there are filenames in the argument list after the filename currently referenced, and the
89085 last command was not a **quit**, **wq**, **xit**, or **ZZ** (see [Exit](#), on page 3410) command, it shall be
89086 an error.

89087 Otherwise, terminate the editing session.

89088 Read

89089 *Synopsis*: `[laddr] r[ead] [!][file]`

89090 If '!' is not the first non-<blank> to follow the command name, a copy of the specified file shall
89091 be appended into the edit buffer after the specified line; line zero specifies that the copy shall be
89092 placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If
89093 no *file* is named, the current pathname shall be the default. If there is no current pathname, then
89094 *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be
89095 an error. Specifying a *file* that is not of type regular shall have unspecified results.

89096 Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and
89097 '!' characters expanded as described in [Command Line Parsing in ex](#) (on page 2705).

89098 The *ex* utility shall then pass two arguments to the program named by the shell edit option; the

89099 first shall be `-c` and the second shall be the expanded arguments to the **read** command as a
 89100 single argument. The standard input of the program shall be set to the standard input of the *ex*
 89101 program when it was invoked. The standard error and standard output of the program shall be
 89102 appended into the edit buffer after the specified line.

89103 Each line in the copied file or program output (as delimited by `<newline>` characters or the end
 89104 of the file or output if it is not immediately preceded by a `<newline>`), shall be a separate line in
 89105 the edit buffer. Any occurrences of `<carriage-return>` and `<newline>` pairs in the output shall be
 89106 treated as single `<newline>` characters.

89107 The special meaning of the `'!'` following the **read** command can be overridden by escaping it
 89108 with a `<backslash>` character.

89109 *Current line*: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual
 89110 mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into
 89111 the edit buffer.

89112 *Current column*: Set to non-`<blank>`.

89113 **Recover**

89114 *Synopsis*: `rec[over][!] file`

89115 If no `'!'` is appended to the command name, and the edit buffer has been modified since the
 89116 last complete write, it shall be an error.

89117 If no *file* operand is specified, then the current pathname shall be used. If there is no current
 89118 pathname or *file* operand, it shall be an error.

89119 If no recovery information has previously been saved about *file*, the **recover** command shall
 89120 behave identically to the **edit** command, and an informational message to this effect shall be
 89121 written.

89122 Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer
 89123 with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the
 89124 one most recently saved shall be recovered, and an informational message that there are
 89125 previous versions of the file that can be recovered shall be written. The editor shall behave as if
 89126 the contents of the edit buffer have already been modified.

89127 *Current file*: Set as described for the **edit** command.

89128 *Current column*: Set as described for the **edit** command.

89129 **Rewind**

89130 *Synopsis*: `rew[ind][!]`

89131 If no `'!'` is appended to the command name, and the edit buffer has been modified since the
 89132 last complete write, it shall be an error, unless the file is successfully written as specified by the
 89133 **autowrite** option.

89134 If the argument list is empty, it shall be an error.

89135 The current argument list reference and the current pathname shall be set to the first filename in
 89136 the argument list.

89137 Replace the contents of the edit buffer with the contents of the file named by the current
 89138 pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be
 89139 empty.

89140 This command shall be affected by the **autowrite** and **writeany** edit options.

89141 *Current line*: Set as described for the **edit** command.

89142 *Current column*: Set as described for the **edit** command.

89143 **Set**

89144 *Synopsis*: `se[t] [option]=[value] ...][nooption ...][option? ...][all]`

89145 When no arguments are specified, write the value of the **term** edit option and those options
89146 whose values have been changed from the default settings; when the argument *all* is specified,
89147 write all of the option values.

89148 Giving an option name followed by the character '?' shall cause the current value of that
89149 option to be written. The '?' can be separated from the option name by zero or more <blank>
89150 characters. The '?' shall be necessary only for Boolean valued options. Boolean options can be
89151 given values by the form **set option** to turn them on or **set nooption** to turn them off; string and
89152 numeric options can be assigned by the form **set option=value**. Any <blank> characters in strings
89153 can be included as is by preceding each <blank> with an escaping <backslash>. More than one
89154 option can be set or listed by a single set command by specifying multiple arguments, each
89155 separated from the next by one or more <blank> characters.

89156 See [Edit Options in ex](#) (on page 2735) for details about specific options.

89157 *Current line*: Unchanged.

89158 *Current column*: Unchanged.

89159 **Shell**

89160 *Synopsis*: `sh[ell]`

89161 Invoke the program named in the **shell** edit option with the single argument **-i** (interactive
89162 mode). Editing shall be resumed when the program exits.

89163 *Current line*: Unchanged.

89164 *Current column*: Unchanged.

89165 **Source**

89166 *Synopsis*: `so[urce] file`

89167 Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

89168 *Current line*: As specified for the individual *ex* commands.

89169 *Current column*: As specified for the individual *ex* commands.

89170 **Substitute**

89171 *Synopsis*: `[2addr] s[substitute] [/pattern/repl/ [options] [count] [flags]]`

89172 `[2addr] & [options] [count] [flags]]`

89173 `[2addr] ~ [options] [count] [flags]]`

89174 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See
89175 [Regular Expressions in ex](#) (on page 2734) and [Replacement Strings in ex](#) (on page 2735).) Any
89176 non-alphabetic, non-<blank> delimiter other than <backslash>, '|', <newline>, or double-
89177 quote can be used instead of '/'. <backslash> characters can be used to escape delimiters,
89178 <backslash> characters, and other special characters.

89179 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If
89180 both *pattern* and *repl* are not specified or are empty (for example, `"/"/`), the last `s` command
89181 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in
89182 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be
89183 replaced by nothing. If the entire replacement pattern is `'%'`, the last replacement pattern to an
89184 `s` command shall be used.

89185 Entering a `<carriage-return>` in *repl* (which requires an escaping `<backslash>` in *ex* mode and an
89186 escaping `<control>-V` in open or *vi* mode) shall split the line at that point, creating a new line in
89187 the edit buffer. The `<carriage-return>` shall be discarded.

89188 If *options* includes the letter `'g'` (**global**), all non-overlapping instances of the pattern in the line
89189 shall be replaced.

89190 If *options* includes the letter `'c'` (**confirm**), then before each substitution the line shall be written;
89191 the written line shall reflect all previous substitutions. On the following line, `<space>` characters
89192 shall be written beneath the characters from the line that are before the *pattern* to be replaced,
89193 and `'^'` characters written beneath the characters included in the *pattern* to be replaced. The *ex*
89194 utility shall then wait for a response from the user. An affirmative response shall cause the
89195 substitution to be done, while any other input shall not make the substitution. An affirmative
89196 response shall consist of a line with the affirmative response (as defined by the current locale) at
89197 the beginning of the line. This line shall be subject to editing in the same way as the *ex* command
89198 line.

89199 If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the
89200 user shall be preserved in the edit buffer after the interrupt.

89201 If the remembered search direction is not set, the `s` command shall set it to forward.

89202 In the second Synopsis, the `&` command shall repeat the previous substitution, as if the `&`
89203 command were replaced by:

```
89204 s/pattern/repl/
```

89205 where *pattern* and *repl* are as specified in the previous `s`, `&`, or `~` command.

89206 In the third Synopsis, the `~` command shall repeat the previous substitution, as if the `'~'` were
89207 replaced by:

```
89208 s/pattern/repl/
```

89209 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from
89210 the previous substitution (including `&` and `~`) command.

89211 These commands shall be affected by the `LC_MESSAGES` environment variable.

89212 *Current line*: Set to the last line in which a substitution occurred, or, unchanged if no substitution
89213 occurred.

89214 *Current column*: Set to non-`<blank>`.

89215 **Suspend**89216 *Synopsis:* `su[suspend] [!]`89217 `st[op] [!]`

89218 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the
 89219 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell
 89220 (see the description of *set -m*).

89221 These commands shall be affected by the **autowrite** and **writeany** edit options.

89222 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

89223 **Tag**89224 *Synopsis:* `ta[g] [!] tagstring`

89225 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see
 89226 *ctags*) description.

89227 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in
 89228 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from
 89229 beginning to end. If no reference is found, it shall be an error and an error message to this effect
 89230 shall be written. If the reference is not found, or if an error occurs while processing a file referred
 89231 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first
 89232 occurrence of such an error.

89233 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression
 89234 used in the editor; for example, for the purposes of the **s** command.

89235 If the *tagstring* is in a file with a different name than the current pathname, set the current
 89236 pathname to the name of that file, and replace the contents of the edit buffer with the contents of
 89237 that file. In this case, if no '!' is appended to the command name, and the edit buffer has been
 89238 modified since the last complete write, it shall be an error, unless the file is successfully written
 89239 as specified by the **autowrite** option.

89240 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

89241 *Current line:* If the tags file contained a line number, set to that line number. If the line number is
 89242 larger than the last line in the edit buffer, an error message shall be written and the current line
 89243 shall be set as specified for the **edit** command.

89244 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no
 89245 matching pattern is found, an error message shall be written and the current line shall be set as
 89246 specified for the **edit** command.

89247 *Current column:* If the tags file contained a line-number reference and that line-number was not
 89248 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern
 89249 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

89250 **Unabbreviate**89251 *Synopsis:* una[bbrev] lhs89252 If *lhs* is not an entry in the current list of abbreviations (see [Abbreviate](#), on page 2713), it shall be
89253 an error. Otherwise, delete *lhs* from the list of abbreviations.89254 *Current line:* Unchanged.89255 *Current column:* Unchanged.89256 **Undo**89257 *Synopsis:* u[ndo]89258 Reverse the changes made by the last command that modified the contents of the edit buffer,
89259 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands
89260 resulting from buffer executions and mapped character expansions, are considered single
89261 commands.89262 If no action that can be undone preceded the **undo** command, it shall be an error.89263 If the **undo** command restores lines that were marked, the mark shall also be restored unless it
89264 was reset subsequent to the deletion of the lines.89265 *Current line:*

- 89266 1. If lines are added or changed in the file, set to the first line added or changed.
-
- 89267 2. Set to the line before the first line deleted, if it exists.
-
- 89268 3. Set to 1 if the edit buffer is not empty.
-
- 89269 4. Set to zero.

89270 *Current column:* Set to non-<blank>.89271 **Unmap**89272 *Synopsis:* unm[ap][!] lhs89273 If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode
89274 map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map
89275 definitions.89276 If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command
89277 mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode
89278 map definitions.89279 *Current line:* Unchanged.89280 *Current column:* Unchanged.

89281 **Version**89282 *Synopsis:* `ve[rsion]`89283 Write a message containing version information for the editor. The format of the message is
89284 unspecified.89285 *Current line:* Unchanged.89286 *Current column:* Unchanged.89287 **Visual**89288 *Synopsis:* `[laddr] vi[sual] [type] [count] [flags]`89289 If *ex* is currently in open or visual mode, the *Synopsis* and behavior of the visual command shall
89290 be the same as the **edit** command, as specified by **Edit** (on page 2715).89291 Otherwise, this command need not be supported on block-mode terminals or terminals with
89292 insufficient capabilities. If standard input, standard output, or standard error are not terminal
89293 devices, the results are unspecified.89294 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
89295 **window**, on page 2742). If the '^' type character was also specified, the **window** edit option
89296 shall be set before being used by the type character.89297 Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type*
89298 shall cause the following effects:

89299 + Place the beginning of the specified line at the top of the display.

89300 - Place the end of the specified line at the bottom of the display.

89301 . Place the beginning of the specified line in the middle of the display.

89302 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line
89303 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place
89304 the beginning of this line as close to the bottom of the displayed lines as possible, while still
89305 displaying the value of the **window** edit option number of lines.89306 *Current line:* Set to the specified line.89307 *Current column:* Set to non-<blank>.89308 **Write**89309 *Synopsis:* `[2addr] w[rite] [!] [>>] [file]`89310 `[2addr] w[rite] [!] [file]`89311 `[2addr] wq [!] [>>] [file]`

89312 If no lines are specified, the lines shall default to the entire file.

89313 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**
89314 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the
89315 **quit** shall not be attempted.89316 If the command name is not followed by one or more <blank> characters, or *file* is not preceded
89317 by a '!' character, the **write** shall be to a file.89318 1. If the >> argument is specified, and the file already exists, the lines shall be appended to
89319 the file instead of replacing its contents. If the >> argument is specified, and the file does
89320 not already exist, it is unspecified whether the write shall proceed as if the >> argument

- 89321 had not been specified or if the write shall fail.
- 89322 2. If the **readonly** edit option is set (see [readonly](#), on page 2739), the **write** shall fail.
- 89323 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
- 89324 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,
89325 the **write** command shall fail.
- 89326 5. If the current pathname is used, and the current pathname has been changed by the **file**
89327 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,
89328 subsequent **writes** shall not fail for this reason (unless the current pathname is changed
89329 again).
- 89330 6. If the whole edit buffer is not being written, and the file to be written exists, the **write**
89331 shall fail.

89332 For rules 1., 2., 3., and 5., the **write** can be forced by appending the character '!' to the
89333 command name.

89334 For rules 2., 3., and 5., the **write** can be forced by setting the **writeany** edit option.

89335 Additional, implementation-defined tests may cause the **write** to fail.

89336 If the edit buffer is empty, a file without any contents shall be written.

89337 An informational message shall be written noting the number of lines and bytes written.

89338 Otherwise, if the command is followed by one or more <blank> characters, and the file is
89339 preceded by '!', the rest of the line after the '!' shall have '%', '#', and '!' characters
89340 expanded as described in [Command Line Parsing in ex](#) (on page 2705).

89341 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the
89342 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a
89343 single argument. The specified lines shall be written to the standard input of the command. The
89344 standard error and standard output of the program, if any, shall be written as described for the
89345 **print** command. If the last character in that output is not a <newline>, a <newline> shall be
89346 written at the end of the output.

89347 The special meaning of the '!' following the **write** command can be overridden by escaping it
89348 with a <backslash> character.

89349 *Current line*: Unchanged.

89350 *Current column*: Unchanged.

89351 **Write and Exit**

89352 *Synopsis*: `[2addr] x[it][!][file]`

89353 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to
89354 the **quit** command, or if a '!' is appended to the command name, to **quit!**.

89355 Otherwise, **xit** shall be equivalent to the **wq** command, or if a '!' is appended to the command
89356 name, to **wq!**.

89357 *Current line*: Unchanged.

89358 *Current column*: Unchanged.

89359 **Yank**89360 *Synopsis:* `[laddr] ya[nk] [buffer] [count]`89361 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall
89362 become a line-mode buffer.89363 *Current line:* Unchanged.89364 *Current column:* Unchanged.89365 **Adjust Window**89366 *Synopsis:* `[laddr] z[!][type . . .] [count] [flags]`89367 If no line is specified, the current line shall be the default; if *type* is omitted as well, the current
89368 line value shall first be incremented by 1. If incrementing the current line would cause it to be
89369 greater than the last line in the edit buffer, it shall be an error.89370 If there are <blank> characters between the *type* argument and the preceding z command name
89371 or optional '!' character, it shall be an error.89372 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
89373 [window](#), on page 2742). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit
89374 option, or if ! was specified, the number of lines in the display minus 1.89375 If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise,
89376 *count* lines starting with the line specified by the *type* argument shall be written.89377 The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

89378 – The specified line shall be decremented by the following value:

89379 $((\text{number of '-' characters}) \times \text{count}) - 1$ 89380 If the calculation would result in a number less than 1, it shall be an error. Write lines from
89381 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
89382 buffer has been written.

89383 + The specified line shall be incremented by the following value:

89384 $((\text{number of '+' characters}) - 1) \times \text{count} + 1$ 89385 If the calculation would result in a number greater than the last line in the edit buffer, it
89386 shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count*
89387 lines or the last line in the edit buffer has been written.89388 =, . If more than a single ' .' or '=' is specified, it shall be an error. The following steps shall
89389 be taken:89390 1. If *count* is zero, nothing shall be written.89391 2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count*
89392 or '!' was specified, *N* shall be:89393 $(\text{count} - 1) / 2$ 89394 Otherwise, *N* shall be:89395 $(\text{count} - 3) / 2$ 89396 If *N* is a number less than 3, no lines shall be written.

- 89397 3. If '=' was specified as the type character, write a line consisting of the smaller of the
89398 number of columns in the display divided by two, or 40 '-' characters.
- 89399 4. Write the current line.
- 89400 5. Repeat step 3.
- 89401 6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall
89402 be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count*
89403 is less than 3, no lines shall be written.

89404 ^ The specified line shall be decremented by the following value:

89405 $((\text{number of '^' characters}) + 1) \times \text{count} - 1$

89406 If the calculation would result in a number less than 1, it shall be an error. Write lines from
89407 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
89408 buffer has been written.

89409 *Current line*: Set to the last line written, unless the type is =, in which case, set to the specified
89410 line.

89411 *Current column*: Set to non-<blank>.

89412 Escape

89413 *Synopsis*: ! *command*
89414 [*addr*] ! *command*

89415 The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as
89416 described in [Command Line Parsing in ex](#) (on page 2705). If the expansion causes the text of the
89417 line to change, it shall be redisplayed, preceded by a single '!' character.

89418 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two
89419 arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments
89420 to the ! command as a single argument.

89421 If no lines are specified, the standard input, standard output, and standard error of the program
89422 shall be set to the standard input, standard output, and standard error of the *ex* program when it
89423 was invoked. In addition, a warning message shall be written if the edit buffer has been
89424 modified since the last complete write, and the **warn** edit option is set.

89425 If lines are specified, they shall be passed to the program as standard input, and the standard
89426 output and standard error of the program shall replace those lines in the edit buffer. Each line in
89427 the program output (as delimited by <newline> characters or the end of the output if it is not
89428 immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any
89429 occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single
89430 <newline> characters. The specified lines shall be copied into the unnamed buffer before they
89431 are replaced, and the unnamed buffer shall become a line-mode buffer.

89432 If in *ex* mode, a single '!' character shall be written when the program completes.

89433 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this
89434 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this
89435 command shall be affected by the **autoprint** edit option.

89436 *Current line:*

- 89437 1. If no lines are specified, unchanged.
- 89438 2. Otherwise, set to the last line read in, if any lines are read in.
- 89439 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
- 89440 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
- 89441 5. Otherwise, set to zero.

89442 *Current column:* If no lines are specified, unchanged. Otherwise, set to non-<blank>.

89443 **Shift Left**

89444 *Synopsis:* [*2addr*] <[< . . .] [*count*] [*flags*]

89445 Shift the specified lines to the start of the line; the number of column positions to be shifted shall
89446 be the number of command characters times the value of the **shiftwidth** edit option. Only
89447 leading <blank> characters shall be deleted or changed into other <blank> characters in shifting;
89448 other characters shall not be affected.

89449 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
89450 buffer.

89451 This command shall be affected by the **autoprint** edit option.

89452 *Current line:* Set to the last line in the lines specified.

89453 *Current column:* Set to non-<blank>.

89454 **Shift Right**

89455 *Synopsis:* [*2addr*] >[> . . .] [*count*] [*flags*]

89456 Shift the specified lines away from the start of the line; the number of column positions to be
89457 shifted shall be the number of command characters times the value of the **shiftwidth** edit
89458 option. The shift shall be accomplished by adding <blank> characters as a prefix to the line or
89459 changing leading <blank> characters into other <blank> characters. Empty lines shall not be
89460 changed.

89461 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
89462 buffer.

89463 This command shall be affected by the **autoprint** edit option.

89464 *Current line:* Set to the last line in the lines specified.

89465 *Current column:* Set to non-<blank>.

89466 **<control>-D**

89467 *Synopsis:* <control>-D

89468 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the
89469 number of lines after the current line in the edit buffer. If the current line is the last line of the
89470 edit buffer it shall be an error.

89471 *Current line:* Set to the last line written.

89472 *Current column:* Set to non-<blank>.

89473 **Write Line Number**89474 *Synopsis:* `[laddr] = [flags]`89475 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of
89476 the specified line.89477 *Current line:* Unchanged.89478 *Current column:* Unchanged.89479 **Execute**89480 *Synopsis:* `[2addr] @ buffer`89481 `[2addr] * buffer`89482 If no *buffer* is specified or is specified as '@' or '*', the last buffer executed shall be used. If no
89483 previous buffer has been executed, it shall be an error.89484 For each line specified by the addresses, set the current line ('.') to the specified line, and
89485 execute the contents of the named *buffer* (as they were at the time the @ command was executed)
89486 as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode
89487 buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.89488 If an error occurs during this process, or a line specified by the addresses does not exist when
89489 the current line would be set to it, or more than a single line was specified by the addresses, and
89490 the contents of the edit buffer are replaced (for example, by the *ex* :edit command) an error
89491 message shall be written, and no more commands resulting from the execution of this command
89492 shall be processed.89493 *Current line:* As specified for the individual *ex* commands.89494 *Current column:* As specified for the individual *ex* commands.89495 **Regular Expressions in ex**89496 The *ex* utility shall support regular expressions that are a superset of the basic regular
89497 expressions described in XBD [Section 9.3](#) (on page 183). A null regular expression ("/") shall
89498 be equivalent to the last regular expression encountered.89499 Regular expressions can be used in addresses to specify lines and, in some commands (for
89500 example, the **substitute** command), to specify portions of a line to be substituted.

89501 The following constructs can be used to enhance the basic regular expressions:

89502 \< Match the beginning of a *word*. (See the definition of *word* at the beginning of [Command](#)
89503 [Descriptions in ex](#) (on page 2710).)89504 \> Match the end of a *word*.89505 ~ Match the replacement part of the last **substitute** command. The <tilde> ('~') character can
89506 be escaped in a regular expression to become a normal character with no special meaning.
89507 The <backslash> shall be discarded.89508 When the editor option **magic** is not set, the only characters with special meanings shall be '^'
89509 at the beginning of a pattern, '\$' at the end of a pattern, and <backslash>. The characters '.',
89510 '*', '[', and '~' shall be treated as ordinary characters unless preceded by a <backslash>;
89511 when preceded by a <backslash> they shall regain their special meaning, or in the case of
89512 <backslash>, be handled as a single <backslash>. <backslash> characters used to escape other
89513 characters shall be discarded.

89514 **Replacement Strings in ex**

89515 The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall
 89516 stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not
 89517 set) shall be replaced by the replacement part of the previous **substitute** command. The
 89518 sequence '\n', where *n* is an integer, shall be replaced by the text matched by the
 89519 corresponding back-reference expression. If the corresponding back-reference expression does
 89520 not match, then the characters '\n' shall be replaced by the empty string.

89521 The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the
 89522 replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause
 89523 the character that follows to be converted to lowercase (uppercase). The string '\L' ('\U') shall
 89524 cause all characters subsequent to it to be converted to lowercase (uppercase) as they are
 89525 inserted by the substitution until the string '\e' or '\E', or the end of the replacement string,
 89526 is encountered.

89527 Otherwise, any character following a <backslash> shall be treated as that literal character, and
 89528 the escaping <backslash> shall be discarded.

89529 An example of case conversion with the **s** command is as follows:

```
89530 :p
89531 The cat sat on the mat.
89532 :s/\<.at\>/\u&/gp
89533 The Cat Sat on the Mat.
89534 :s/S\(.*\)M/S\U\l\eM/p
89535 The Cat SAT ON THE Mat.
```

89536 **Edit Options in ex**

89537 The *ex* utility has a number of options that modify its behavior. These options have default
 89538 settings, which can be changed using the **set** command.

89539 Options are Boolean unless otherwise specified.

89540 **autoindent, ai**

89541 [Default *unset*]

89542 If **autoindent** is set, each line in input mode shall be indented (using first as many <tab>
 89543 characters as possible, as determined by the editor option **tabstop**, and then using <space>
 89544 characters) to align with another line, as follows:

- 89545 1. If in open or visual mode and the text input is part of a line-oriented command (see the
 89546 EXTENDED DESCRIPTION in *vi*), align to the first column.
- 89547 2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
 - 89548 a. If a line was previously inserted as part of this command, it shall be set to the
 89549 indentation of the last inserted line by default, or as otherwise specified for the
 89550 <control>-D character in **Input Mode Commands in vi** (on page 3410).
 - 89551 b. Otherwise, it shall be set to the indentation of the previous current line, if any;
 89552 otherwise, to the first column.
- 89553 3. For the *ex* **a**, **i**, and **c** commands, indentation for each line shall be set as follows:

- 89554 a. If a line was previously inserted as part of this command, it shall be set to the
89555 indentation of the last inserted line by default, or as otherwise specified for the *eof*
89556 character in [Scroll](#) (on page 2709).
- 89557 b. Otherwise, if the command is the *ex a* command, it shall be set to the line
89558 appended after, if any; otherwise to the first column.
- 89559 c. Otherwise, if the command is the *ex i* command, it shall be set to the line inserted
89560 before, if any; otherwise to the first column.
- 89561 d. Otherwise, if the command is the *ex c* command, it shall be set to the indentation of
89562 the line replaced.

89563 **autoprint, ap**

89564 [Default *set*]

89565 If **autoprint** is set, the current line shall be written after each *ex* command that modifies the
89566 contents of the current edit buffer, and after each **tag** command for which the tag search pattern
89567 was found or tag line number was valid, unless:

- 89568 1. The command was executed while in open or visual mode.
- 89569 2. The command was executed as part of a **global** or **v** command or **@** buffer execution.
- 89570 3. The command was the form of the **read** command that reads a file into the edit buffer.
- 89571 4. The command was the **append**, **change**, or **insert** command.
- 89572 5. The command was not terminated by a <newline>.
- 89573 6. The current line shall be written by a flag specified to the command; for example, **delete #**
89574 shall write the current line as specified for the flag modifier to the **delete** command, and
89575 not as specified by the **autoprint** edit option.

89576 **autowrite, aw**

89577 [Default *unset*]

89578 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to
89579 any file, the contents of the edit buffer shall be written as if the *ex write* command had been
89580 specified without arguments, before each command affected by the **autowrite** edit option is
89581 executed. Appending the character '!' to the command name of any of the *ex* commands
89582 except '!' shall prevent the write. If the write fails, it shall be an error and the command shall
89583 not be executed.

89584 **beautify, bf**

89585 XSI [Default *unset*]

89586 If **beautify** is set, all non-printable characters, other than <tab>, <newline>, and <form-feed>
89587 characters, shall be discarded from text read in from files.

- 89588 **directory, dir**
89589 [Default *implementation-defined*]
89590 The value of this option specifies the directory in which the editor buffer is to be placed. If this
89591 directory is not writable by the user, the editor shall quit.
- 89592 **edcompatible, ed**
89593 [Default *unset*]
89594 Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled
89595 by repeating the suffixes.
- 89596 **errorbells, eb**
89597 [Default *unset*]
89598 If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse
89599 video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.
- 89600 **exrc**
89601 [Default *unset*]
89602 If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in [Initialization in](#)
89603 [ex and vi](#) (on page 2701). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory
89604 during initialization, unless the current directory is that named by the *HOME* environment
89605 variable.
- 89606 **ignorecase, ic**
89607 [Default *unset*]
89608 If **ignorecase** is set, characters that have uppercase and lowercase representations shall have
89609 those representations considered as equivalent for purposes of regular expression comparison.
89610 The **ignorecase** edit option shall affect all remembered regular expressions; for example,
89611 unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the
89612 last basic regular expression in a case-sensitive fashion.
- 89613 **list**
89614 [Default *unset*]
89615 If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for
89616 the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall
89617 be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual
89618 text input mode, when the cursor does not rest on any character in the line, it shall rest on the
89619 ' \$ ' marking the end of the line.

89620 **magic**89621 [Default *set*]

89622 If **magic** is set, modify the interpretation of characters in regular expressions and substitution
 89623 replacement strings (see [Regular Expressions in ex](#) (on page 2734) and [Replacement Strings in](#)
 89624 [ex](#), on page 2735).

89625 **mesg**89626 [Default *set*]

89627 If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the
 89628 terminal shall be turned on while in open or visual mode. The shell-level command *mesg n* shall
 89629 take precedence over any setting of the *ex mesg* option; that is, if **mesg y** was issued before the
 89630 editor started (or in a shell escape), such as:

89631 `:!mesg y`

89632 the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable
 89633 incoming messages if **mesg n** was issued.

89634 **number, nu**89635 [Default *unset*]

89636 If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line
 89637 numbers, in the format specified by the **print** command with the **#** flag specified. In *ex* text input
 89638 mode, each line shall be preceded by the line number it will have in the file.

89639 In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in
 89640 the format specified by the *ex print* command with the **#** flag specified. This line number shall
 89641 not be considered part of the line for the purposes of evaluating the current column; that is,
 89642 column position 1 shall be the first column position after the format specified by the **print**
 89643 command.

89644 **paragraphs, para**89645 [Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

89646 The **paragraphs** edit option shall define additional paragraph boundaries for the open and
 89647 visual mode commands. The **paragraphs** edit option can be set to a character string consisting of
 89648 zero or more character pairs. It shall be an error to set it to an odd number of characters.

89649 **prompt**89650 [Default *set*]

89651 If **prompt** is set, *ex* command mode input shall be prompted for with a <colon> (':'); when
 89652 unset, no prompt shall be written.

89653 **readonly**89654 [Default *see text*]

89655 If the **readonly** edit option is set, read-only mode shall be enabled (see **Write**, on page 2729). The
 89656 **readonly** edit option shall be initialized to set if either of the following conditions are true:

89657 The command-line option **-R** was specified.

89658 Performing actions equivalent to the *access()* function called with the following arguments
 89659 indicates that the file lacks write permission:

- 89660 1. The current pathname is used as the *path* argument.
- 89661 2. The constant **W_OK** is used as the *amode* argument.

89662 The **readonly** edit option may be initialized to set for other, implementation-defined reasons.
 89663 The **readonly** edit option shall not be initialized to unset based on any special privileges of the
 89664 user or process. The **readonly** edit option shall be reinitialized each time that the contents of the
 89665 edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly
 89666 set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again
 89667 be reinitialized each time that the contents of the edit buffer are replaced.

89668 **redraw**89669 [Default *unset*]

89670 The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a
 89671 large amount of output to the terminal, it is useful only at high transmission speeds.)

89672 **remap**89673 [Default *set*]

89674 If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation
 89675 shall continue until a final product is obtained. If unset, only a one-step translation shall be
 89676 done.

89677 **report**

89678 [Default 5]

89679 The value of this **report** edit option specifies what number of lines being added, copied, deleted,
 89680 or modified in the edit buffer will cause an informational message to be written to the user. The
 89681 following conditions shall cause an informational message. The message shall contain the
 89682 number of lines added, copied, deleted, or modified, but is otherwise unspecified.

89683 An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the
 89684 value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v**
 89685 command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.

89686 An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option
 89687 plus 1 number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi*
 89688 buffer execution, shall cause an informational message to be written.

89689 An *ex* **global**, **v**, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or
 89690 deletes a total of at least the value of the **report** edit option number of lines, and which is
 89691 not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an
 89692 informational message to be written. (For example, if 3 lines were added and 8 lines
 89693 deleted during an *ex* **visual** command, 5 would be the number compared against the

89694 **report** edit option after the command completed.)

89695 **scroll, scr**

89696 [Default (number of lines in the display -1)/2]

89697 The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex*
89698 <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the
89699 initial number of lines to scroll when no previous <control>-D or <control>-U command has
89700 been executed.

89701 **sections**

89702 [Default in the POSIX locale `NHSHH HUnhsh`]

89703 The **sections** edit option shall define additional section boundaries for the open and visual mode
89704 commands. The **sections** edit option can be set to a character string consisting of zero or more
89705 character pairs; it shall be an error to set it to an odd number of characters.

89706 **shell, sh**

89707 [Default from the environment variable `SHELL`]

89708 The value of this option shall be a string. The default shall be taken from the `SHELL`
89709 environment variable. If the `SHELL` environment variable is null or empty, the *sh* (see *sh*) utility
89710 shall be the default.

89711 **shiftwidth, sw**

89712 [Default 8]

89713 The value of this option shall give the width in columns of an indentation level used during
89714 autoindentation and by the shift commands (< and >).

89715 **showmatch, sm**

89716 [Default *unset*]

89717 The functionality described for the **showmatch** edit option need not be supported on block-
89718 mode terminals or terminals with insufficient capabilities.

89719 If **showmatch** is set, in open or visual mode, when a ') ' or ' } ' is typed, if the matching ' (' or
89720 ' { ' is currently visible on the display, the matching ' (' or ' { ' shall be flagged moving the
89721 cursor to its location for an unspecified amount of time.

89722 **showmode**

89723 [Default *unset*]

89724 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be
89725 displayed on the last line of the display. Command mode and text input mode shall be
89726 differentiated; other unspecified modes and implementation-defined information may be
89727 displayed.

- 89728 **slowopen**
89729 [Default *unset*]
89730 If **slowopen** is set during open and visual text input modes, the editor shall not update portions
89731 of the display other than those display line columns that display the characters entered by the
89732 user (see [Input Mode Commands in vi](#), on page 3410).
- 89733 **tabstop, ts**
89734 [Default 8]
89735 The value of this edit option shall specify the column boundary used by a <tab> in the display
89736 (see [autoprint, ap](#) (on page 2736) and [Input Mode Commands in vi](#), on page 3410).
- 89737 **taglength, tl**
89738 [Default zero]
89739 The value of this edit option shall specify the maximum number of characters that are
89740 considered significant in the user-specified tag name and in the tag name from the tags file. If
89741 the value is zero, all characters in both tag names shall be significant.
- 89742 **tags**
89743 [Default *see text*]
89744 The value of this edit option shall be a string of <blank>-delimited pathnames of files used by
89745 the **tag** command. The default value is unspecified.
- 89746 **term**
89747 [Default from the environment variable *TERM*]
89748 The value of this edit option shall be a string. The default shall be taken from the *TERM* variable
89749 in the environment. If the *TERM* environment variable is empty or null, the default is
89750 unspecified. The editor shall use the value of this edit option to determine the type of the display
89751 device.
89752 The results are unspecified if the user changes the value of the term edit option after editor
89753 initialization.
- 89754 **terse**
89755 [Default *unset*]
89756 If **terse** is set, error messages may be less verbose. However, except for this caveat, error
89757 messages are unspecified. Furthermore, not all error messages need change for different settings
89758 of this option.

warn

89759 [Default *set*]

89761 If **warn** is set, and the contents of the edit buffer have been modified since they were last
89762 completely written, the editor shall write a warning message before certain ! commands (see
89763 [Escape](#), on page 2732).

window

89764 [Default *see text*]

89766 A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in
89767 visual mode, to specify the number of lines displayed when the screen is repainted.

89768 If the **-w** command-line option is not specified, the default value shall be set to the value of the
89769 *LINES* environment variable. If the *LINES* environment variable is empty or null, the default
89770 shall be the number of lines in the display minus 1.

89771 Setting the **window** edit option to zero or to a value greater than the number of lines in the
89772 display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable)
89773 shall cause the **window** edit option to be set to the number of lines in the display minus 1.

89774 The baud rate of the terminal line may change the default in an implementation-defined manner.

wrapmargin, wm

89775 [Default 0]

89777 If the value of this edit option is zero, it shall have no effect.

89778 If not in the POSIX locale, the effect of this edit option is implementation-defined.

89779 Otherwise, it shall specify a number of columns from the ending margin of the terminal.

89780 During open and visual text input modes, for each character for which any part of the character
89781 is displayed in a column that is less than **wrapmargin** columns from the ending margin of the
89782 display line, the editor shall behave as follows:

89783 1. If the character triggering this event is a <blank>, it, and all immediately preceding
89784 <blank> characters on the current line entered during the execution of the current text
89785 input command, shall be discarded, and the editor shall behave as if the user had entered
89786 a single <newline> instead. In addition, if the next user-entered character is a <space>, it
89787 shall be discarded as well.

89788 2. Otherwise, if there are one or more <blank> characters on the current line immediately
89789 preceding the last group of inserted non-<blank> characters which was entered during
89790 the execution of the current text input command, the <blank> characters shall be replaced
89791 as if the user had entered a single <newline> instead.

89792 If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any
89793 <blank> characters at or after the cursor in the current line shall be discarded.

89794 The ending margin shall be determined by the system or overridden by the user, as described for
89795 *COLUMNS* in the ENVIRONMENT VARIABLES section and XBD [Chapter 8](#) (on page 173).

89796 **wrapsan, ws**

89797 [Default *set*]

89798 If **wrapsan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, N, and n
89799 commands) shall wrap around the beginning or end of the edit buffer; when unset, searches
89800 shall stop at the beginning or end of the edit buffer.

89801 **writeany, wa**

89802 [Default *unset*]

89803 If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be
89804 inhibited, as described in editor option **autowrite**.

89805 **EXIT STATUS**

89806 The following exit values shall be returned:

89807 0 Successful completion.

89808 >0 An error occurred.

89809 **CONSEQUENCES OF ERRORS**

89810 When any error is encountered and the standard input is not a terminal device file, *ex* shall not
89811 write the file or return to command or text input mode, and shall terminate with a non-zero exit
89812 status.

89813 Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP
89814 asynchronous event.

89815 Otherwise, when an error is encountered, the editor shall behave as specified in [Command Line](#)
89816 [Parsing in ex](#) (on page 2705).

89817 **APPLICATION USAGE**

89818 If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

89819 The **next** command can accept more than one file, so usage such as:

89820 `next `ls [abc]*``

89821 is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect
89822 only one file and unspecified results occur.

89823 **EXAMPLES**

89824 None.

89825 **RATIONALE**

89826 The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V
89827 implementations of *ex* and *vi*.

89828 A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and
89829 rejected for inclusion. Neither option provided the level of security that users might expect.

89830 It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to
89831 implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor
89832 addressing; thus, it is not a mandatory requirement that such features should work on all
89833 terminals. It is the intention, however, that an *ex* implementation should provide the full set of
89834 capabilities on all terminals capable of supporting them.

89835

Options

89836

89837

89838

89839

89840

89841

89842

The `-c` replacement for `+command` was inspired by the `-e` option of `sed`. Historically, all such commands (see `edit` and `next` as well) were executed from the last line of the edit buffer. This meant, for example, that `"/pattern"` would fail unless the `wrapsan` option was set. POSIX.1-2017 requires conformance to historical practice. The `+command` option is no longer specified by POSIX.1-2017 but may be present in some implementations. Historically, some implementations restricted the `ex` commands that could be listed as part of the command line arguments. For consistency, POSIX.1-2017 does not permit these restrictions.

89843

89844

89845

89846

89847

In historical implementations of the editor, the `-R` option (and the `readonly` edit option) only prevented overwriting of files; appending to files was still permitted, mapping loosely into the `cs`h `noclobber` variable. Some implementations, however, have not followed this semantic, and `readonly` does not permit appending either. POSIX.1-2017 follows the latter practice, believing that it is a more obvious and intuitive meaning of `readonly`.

89848

89849

89850

The `-s` option suppresses all interactive user feedback and is useful for editing scripts in batch jobs. The list of specific effects is historical practice. The terminal type `"incapable of supporting open and visual modes"` has historically been named `"dumb"`.

89851

89852

The `-t` option was required because the `ctags` utility appears in POSIX.1-2017 and the option is available in all historical implementations of `ex`.

89853

89854

89855

89856

89857

Historically, the `ex` and `vi` utilities accepted a `-x` option, which did encryption based on the algorithm found in the historical `crypt` utility. The `-x` option for encryption, and the associated `crypt` utility, were omitted because the algorithm used was not specifiable and the export control laws of some nations make it difficult to export cryptographic technology. In addition, it did not historically provide the level of security that users might expect.

89858

Standard Input

89859

89860

An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, `<control>-D`, is historically an `ex` command.

89861

89862

89863

89864

89865

89866

There was no maximum line length in historical implementations of `ex`. Specifically, as it was parsed in chunks, the addresses had a different maximum length than the filenames. Further, the maximum line buffer size was declared as `BUFSIZ`, which was different lengths on different systems. This version selected the value of `{LINE_MAX}` to impose a reasonable restriction on portable usage of `ex` and to aid test suite writers in their development of realistic tests that exercise this limit.

89867

Input Files

89868

89869

89870

89871

89872

89873

89874

It was an explicit decision by the standard developers that a `<newline>` be added to any file lacking one. It was believed that this feature of `ex` and `vi` was relied on by users in order to make text files lacking a trailing `<newline>` more portable. It is recognized that this will require a user-specified option or extension for implementations that permit `ex` and `vi` to edit files of type other than text if such files are not otherwise identified by the system. It was agreed that the ability to edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an `ex` or `vi` implementation be required to handle files other than text files.

89875

89876

89877

89878

89879

The paragraph in the INPUT FILES section, `"By default, ..."`, is intended to close a long-standing security problem in `ex` and `vi`; that of the `"modeline"` or `"modelines"` edit option. This feature allows any line in the first or last five lines of the file containing the strings `"ex:"` or `"vi:"` (and, apparently, `"ei:"` or `"vx:"`) to be a line containing editor commands, and `ex` interprets all the text up to the next `':'` or `<newline>` as a command. Consider the

89880 consequences, for example, of an unsuspecting user using *ex* or *vi* as the editor when replying to
89881 a mail message in which a line such as:

```
89882 ex:! rm -rf :
```

89883 appeared in the signature lines. The standard developers believed strongly that an editor should
89884 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from
89885 their implementations of *ex* and *vi*.

89886 **Asynchronous Events**

89887 The intention of the phrase “complete write” is that the entire edit buffer be written to stable
89888 storage. The note regarding temporary files is intended for implementations that use temporary
89889 files to back edit buffers unnamed by the user.

89890 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual
89891 mode; that is, it exited visual mode and entered *ex* mode. POSIX.1-2017 permits, but does not
89892 require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text input
89893 mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi* alerted the
89894 terminal on this event, and some did not. POSIX.1-2017 requires that SIGINT behave identically
89895 to <ESC>, and that the terminal not be alerted.

89896 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as
89897 completed lines were retained, but any partial line discarded, and the editor returned to
89898 command mode. POSIX.1-2017 is silent on this issue; implementations are encouraged to follow
89899 historical practice, where possible.

89900 Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore
89901 impossible to suspend the editor in visual text input mode. There are two major reasons for this.
89902 The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where
89903 the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if
89904 SIGTSTP was delivered to the process group in the default manner. The second was that most
89905 implementations of the UNIX *curses* package did not handle SIGTSTP safely, and the receipt of
89906 SIGTSTP at the wrong time would cause them to crash. POSIX.1-2017 is silent on this issue;
89907 implementations are encouraged to treat suspension as an asynchronous event if possible.

89908 Historically, modifications to the edit buffer made before SIGINT interrupted an operation were
89909 retained; that is, anywhere from zero to all of the lines to be modified might have been modified
89910 by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT.
89911 POSIX.1-2017 permits this behavior, noting that the **undo** command is required to be able to
89912 undo these partially completed commands.

89913 The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is
89914 unspecified because some implementations attempt to save the edit buffer in a useful state when
89915 other signals are received.

89916 **Standard Error**

89917 For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to
89918 invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination
89919 condition. Diagnostic messages should not be confused with the error messages generated by
89920 inappropriate or illegal user commands.

89921
89922
89923
89924
89925
89926
89927
89928
89929
89930
89931
89932
89933
89934
89935
89936
89937
89938
89939
89940
89941
89942
89943
89944
89945
89946
89947
89948
89949
89950
89951
89952
89953
89954
89955
89956
89957
89958
89959
89960
89961
89962
89963
89964
89965
89966

Initialization in ex and vi

If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the filename argument (the first filename argument in the case of the **next** command) and the alternate pathname will be set to the previous current pathname, if there was one.
2. In the case of the file read/write forms of the **read** and **write** commands, if there is no current pathname, the current pathname will be set to the filename argument.
3. Otherwise, the alternate pathname will be set to the filename argument.

For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands **:write**, **!command**, and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for some reason, the alternate pathname would be set. The **read** and **write** commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the **:source** command. POSIX.1-2017 requires conformance to historical practice. Implementations adding commands that take filenames as arguments are encouraged to set the alternate pathname as described here.

Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed in the *\$HOME* directory. POSIX.1-2017 prohibits this behavior.

Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX system commands inside a **.exrc** file. POSIX.1-2017 does not specify the **sourceany** option, and historical implementations are encouraged to delete it.

The **.exrc** files must be owned by the real ID of the user, and not writable by anyone other than the owner. The appropriate privileges exception is intended to permit users to acquire special privileges, but continue to use the **.exrc** files in their home directories.

System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off, so by default, local **.exrc** files were not read. The problem this was intended to solve was that System V permitted users to give away files, so there is no possible ownership or writeability test to ensure that the file is safe. This is still a security problem on systems where users can give away files, but there is nothing additional that POSIX.1-2017 can do. The implementation-defined exception is intended to permit groups to have local **.exrc** files that are shared by users, by creating pseudo-users to own the shared files.

POSIX.1-2017 does not mention system-wide *ex* and *vi* start-up files. While they exist in several implementations of *ex* and *vi*, they are not present in any implementations considered historical practice by POSIX.1-2017. Implementations that have such files should use them only if they are owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if they are not writable by any user other than their owner. System-wide start-up files should be read before the *EXINIT* variable, **\$HOME/.exrc**, or local **.exrc** files are evaluated.

Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although ones requiring that the edit buffer already contain lines of text generally caused historical implementations of the editor to drop **core**. POSIX.1-2017 requires that any *ex* command be

89967 permitted in the *EXINIT* variable and *.exrc* files, for simplicity of specification and consistency,
89968 although many of them will obviously fail under many circumstances.

89969 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with
89970 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded
89971 during the initialization phase not be lost; that is, loading the edit buffer should fail if the *.exrc*
89972 file read in the contents of a file and did not subsequently write the edit buffer. An additional
89973 intent of this phrase is to specify that the initial current line and column is set as specified for the
89974 individual *ex* commands.

89975 Historically, the *-t* option behaved as if the tag search were a *+command*; that is, it was executed
89976 from the last line of the file specified by the tag. This resulted in the search failing if the pattern
89977 was a forward search pattern and the **wrapsan** edit option was not set. POSIX.1-2017 does not
89978 permit this behavior, requiring that the search for the tag pattern be performed on the entire file,
89979 and, if not found, that the current line be set to a more reasonable location in the file.

89980 Historically, the empty edit buffer presented for editing when a file was not specified by the user
89981 was unnamed. This is permitted by POSIX.1-2017; however, implementations are encouraged to
89982 provide users a temporary filename for this buffer because it permits them the use of *ex*
89983 commands that use the current pathname during temporary edit sessions.

89984 Historically, the file specified using the *-t* option was not part of the current argument list. This
89985 practice is permitted by POSIX.1-2017; however, implementations are encouraged to include its
89986 name in the current argument list for consistency.

89987 Historically, the *-c* command was generally not executed until a file that already exists was
89988 edited. POSIX.1-2017 requires conformance to this historical practice. Commands that could
89989 cause the *-c* command to be executed include the *ex* commands **edit**, **next**, **recover**, **rewind**, and
89990 **tag**, and the *vi* commands *<control>-^* and *<control>-]*. Historically, reading a file into an edit
89991 buffer did not cause the *-c* command to be executed (even though it might set the current
89992 pathname) with the exception that it did cause the *-c* command to be executed if: the editor was
89993 in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read
89994 commands had yet been attempted. For consistency and simplicity of specification,
89995 POSIX.1-2017 does not permit this behavior.

89996 Historically, the *-r* option was the same as a normal edit session if there was no recovery
89997 information available for the file. This allowed users to enter:

```
89998 vi -r *.c
```

89999 and recover whatever files were recoverable. In some implementations, recovery was attempted
90000 only on the first file named, and the file was not entered into the argument list; in others,
90001 recovery was attempted for each file named. In addition, some historical implementations
90002 ignored *-r* if *-t* was specified or did not support command line *file* arguments with the *-t* option.
90003 For consistency and simplicity of specification, POSIX.1-2017 disallows these special cases, and
90004 requires that recovery be attempted the first time each file is edited.

90005 Historically, *vi* initialized the *`* and *'* marks, but *ex* did not. This meant that if the first command
90006 in *ex* mode was **visual** or if an *ex* command was executed first (for example, *vi +10 file*), *vi*
90007 was entered without the marks being initialized. Because the standard developers believed the marks
90008 to be generally useful, and for consistency and simplicity of specification, POSIX.1-2017 requires
90009 that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is
90010 not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however,
90011 it has always been possible to set (and use) marks in empty edit buffers in open and visual mode
90012 edit sessions.

90013 **Addressing**

90014 Historically, *ex* and *vi* accepted the additional addressing forms '\/' and '\?'. They were
 90015 equivalent to "//" and "??", respectively. They are not required by POSIX.1-2017, mostly
 90016 because nobody can remember whether they ever did anything different historically.

90017 Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the %
 90018 address in empty files for others. For consistency, POSIX.1-2017 requires support for the former
 90019 in the few commands where it makes sense, and disallows it otherwise. In addition, because
 90020 POSIX.1-2017 requires that % be logically equivalent to "1,\$", it is also supported where it
 90021 makes sense and disallowed otherwise.

90022 Historically, the % address could not be followed by further addresses. For consistency and
 90023 simplicity of specification, POSIX.1-2017 requires that additional addresses be supported.

90024 All of the following are valid *addresses*:

90025 +++ Three lines after the current line.
 90026 /re/- One line before the next occurrence of *re*.
 90027 -2 Two lines before the current line.
 90028 3 ---- 2 Line one (note intermediate negative address).
 90029 1 2 3 Line six.

90030 Any number of addresses can be provided to commands taking addresses; for example,
 90031 "1,2,3,4,5p" prints lines 4 and 5, because two is the greatest valid number of addresses
 90032 accepted by the **print** command. This, in combination with the <semicolon> delimiter, permits
 90033 users to create commands based on ordered patterns in the file. For example, the command
 90034 **3;/foo/+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next
 90035 two lines. Note that the address **3;** must be evaluated before being discarded because the search
 90036 origin for the **/foo/** command depends on this.

90037 Historically, values could be added to addresses by including them after one or more <blank>
 90038 characters; for example, **3 - 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as
 90039 **/foo/+5**. However, only absolute values could be added; for example, **5 /foo/** was an error.
 90040 POSIX.1-2017 requires conformance to historical practice. Address offsets are separately
 90041 specified from addresses because they could historically be provided to visual mode search
 90042 commands.

90043 Historically, any missing addresses defaulted to the current line. This was true for leading and
 90044 trailing <comma>-delimited addresses, and for trailing <semicolon>-delimited addresses. For
 90045 consistency, POSIX.1-2017 requires it for leading <semicolon> addresses as well.

90046 Historically, *ex* and *vi* accepted the '^' character as both an address and as a flag offset for
 90047 commands. In both cases it was identical to the '-' character. POSIX.1-2017 does not require or
 90048 prohibit this behavior.

90049 Historically, the enhancements to basic regular expressions could be used in addressing; for
 90050 example, '~', '\<', and '\>'. POSIX.1-2017 requires conformance to historical practice; that
 90051 is, that regular expression usage be consistent, and that regular expression enhancements be
 90052 supported wherever regular expressions are used.

90053 **Command Line Parsing in ex**

90054 Historical *ex* command parsing was even more complex than that described here. POSIX.1-2017
 90055 requires the subset of the command parsing that the standard developers believed was
 90056 documented and that users could reasonably be expected to use in a portable fashion, and that
 90057 was historically consistent between implementations. (The discarded functionality is obscure, at
 90058 best.) Historical implementations will require changes in order to comply with POSIX.1-2017;
 90059 however, users are not expected to notice any of these changes. Most of the complexity in *ex*
 90060 parsing is to handle three special termination cases:

- 90061 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by
 90062 <newline> characters (they can contain <vertical-line> characters that are usually shell
 90063 pipes).
- 90064 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,
 90065 optionally containing <vertical-line> characters, as their first arguments.
- 90066 3. The **s** command takes a regular expression as its first argument, and uses the delimiting
 90067 characters to delimit the command.

90068 Historically, <vertical-line> characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and
 90069 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the
 90070 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did
 90071 not delimit the command at all. For example, the following commands are all valid:

```
90072 :edit +25 | s/abc/ABC/ file.c
90073 :s/ | /PIPE/
90074 :read !spell % | columnate
90075 :global/pattern/p | l
90076 :s/a/b/ | s/c/d | set
```

90077 Historically, empty or <blank> filled lines in **.exrc** files and **sourced** files (as well as *EXINIT*
 90078 variables and *ex* command scripts) were treated as default commands; that is, **print** commands.
 90079 POSIX.1-2017 specifically requires that they be ignored when encountered in **.exrc** and **sourced**
 90080 files to eliminate a common source of new user error.

90081 Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were
 90082 handled oddly when executed from *ex* mode. For example, the command `||| <carriage-return>`,
 90083 when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command `|`
 90084 would only display the line after the next line, instead of the next two lines. The former worked
 90085 more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. POSIX.1-2017
 90086 requires the *vi* behavior; that is, a single default command and line number increment for each
 90087 command separator, and trailing <newline> characters after <vertical-line> separators are
 90088 discarded.

90089 Historically, *ex* permitted a single extra <colon> as a leading command character; for example,
 90090 **:g/pattern/:p** was a valid command. POSIX.1-2017 generalizes this to require that any number of
 90091 leading <colon> characters be stripped.

90092 Historically, any prefix of the **delete** command could be followed without intervening <blank>
 90093 characters by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*.
 90094 POSIX.1-2017 requires conformance to historical practice.

90095 Historically, the **k** command could be followed by the mark name without intervening <blank>
 90096 characters. POSIX.1-2017 requires conformance to historical practice.

90097 Historically, the **s** command could be immediately followed by flag and option characters; for
 90098 example, **s/e/E/|s|sgc3p** was a valid command. However, flag characters could not stand alone;

90099 for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would
 90100 succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the
 90101 command.) Another issue was that option characters had to precede flag characters even when
 90102 the command was fully specified; for example, the command **s/e/E/pg** would fail, while the
 90103 command **s/e/E/gp** would succeed. POSIX.1-2017 requires conformance to historical practice.

90104 Historically, the first command name that had a prefix matching the input from the user was the
 90105 executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command.
 90106 Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**.
 90107 POSIX.1-2017 requires conformance to historical practice. The restriction on command search
 90108 order for implementations with extensions is to avoid the addition of commands such that the
 90109 historical prefixes would fail to work portably.

90110 Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands,
 90111 separated by <vertical-line> characters, that entered or exited visual mode or the editor. Because
 90112 implementations of *vi* exist that do not exhibit this failure mode, POSIX.1-2017 does not permit
 90113 it.

90114 The requirement that alphabetic command names consist of all following alphabetic characters
 90115 up to the next non-alphabetic character means that alphabetic command names must be
 90116 separated from their arguments by one or more non-alphabetic characters, normally a <blank>
 90117 or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

90118 Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*,
 90119 <newline>, <carriage-return>) erased any prompting character and displayed the next lines
 90120 without scrolling the terminal; that is, immediately below any previously displayed lines. This
 90121 provided a cleaner presentation of the lines in the file for the user. POSIX.1-2017 does not require
 90122 this behavior because it may be impossible in some situations; however, implementations are
 90123 strongly encouraged to provide this semantic if possible.

90124 Historically, it was possible to change files in the middle of a command, and have the rest of the
 90125 command executed in the new file; for example:

```
90126 :edit +25 file.c | s/abc/ABC/ | 1
```

90127 was a valid command, and the substitution was attempted in the newly edited file.
 90128 POSIX.1-2017 requires conformance to historical practice. The following commands are
 90129 examples that exercise the *ex* parser:

```
90130 echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
90131 vi
```

```
90132 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq
```

90133 Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or *****
 90134 commands changing edit buffers during execution of their associated commands. Because this
 90135 would almost invariably result in catastrophic failure of the editor, and implementations exist
 90136 that do exhibit these problems, POSIX.1-2017 requires that changing the edit buffer during a
 90137 **global** or **v** command, or during a **@** or ***** command for which there will be more than a single
 90138 execution, be an error. Implementations supporting multiple edit buffers simultaneously are
 90139 strongly encouraged to apply the same semantics to switching between buffers as well.

90140 The *ex* command quoting required by POSIX.1-2017 is a superset of the quoting in historical
 90141 implementations of the editor. For example, it was not historically possible to escape a <blank>
 90142 in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had been
 90143 entered for the edit command, and there was no method of escaping a <blank> in the first
 90144 argument of an **edit**, **ex**, **next**, or **visual** command at all. POSIX.1-2017 extends historical
 90145 practice, requiring that quoting behavior be made consistent across all *ex* commands, except for

90146 the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V
90147 instead of <backslash> characters for quoting. For those four commands, POSIX.1-2017 requires
90148 conformance to historical practice.

90149 Backslash quoting in *ex* is non-intuitive. <backslash>-escapes are ignored unless they escape a
90150 special character; for example, when performing *file* argument expansion, the string "\\%" is
90151 equivalent to '\%', not "\<current_pathname>". This can be confusing for users because
90152 <backslash> is usually one of the characters that causes shell expansion to be performed, and
90153 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are
90154 only considered if they escape a special character, and a quoting character must be provided for
90155 each layer of parsing for which the character is special. As another example, only a single
90156 <backslash> is necessary for the '\1' sequence in substitute replacement patterns, because the
90157 character '1' is not special to any parsing layer above it.

90158 <control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands
90159 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character
90160 may be escaped by a <control>-V whether it would have a special meaning or not. POSIX.1-2017
90161 requires conformance to historical practice.

90162 Historical implementations of the editor did not require delimiters within character classes to be
90163 escaped; for example, the command **s/[/]/** on the string "xxx/yyy" would delete the '/' from
90164 the string. POSIX.1-2017 disallows this historical practice for consistency and because it places a
90165 large burden on implementations by requiring that knowledge of regular expressions be built
90166 into the editor parser.

90167 Historically, quoting <newline> characters in *ex* commands was handled inconsistently. In most
90168 cases, the <newline> character always terminated the command, regardless of any preceding
90169 escape character, because <backslash> characters did not escape <newline> characters for most
90170 *ex* commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted
90171 <newline> characters to be escaped (although in the case of **map** and **abbreviation**, <control>-V
90172 characters escaped them instead of <backslash> characters). This was true in not only the
90173 command line, but also **.exrc** and **sourced** files. For example, the command:

```
90174 map = foo<control-V><newline>bar
```

90175 would succeed, although it was sometimes difficult to get the <control>-V and the inserted
90176 <newline> passed to the *ex* parser. For consistency and simplicity of specification, POSIX.1-2017
90177 requires that it be possible to escape <newline> characters in *ex* commands at all times, using
90178 <backslash> characters for most *ex* commands, and using <control>-V characters for the **map**
90179 and **abbreviation** commands. For example, the command **print<newline>list** is required to be
90180 parsed as the single command **print<newline>list**. While this differs from historical practice,
90181 POSIX.1-2017 developers believed it unlikely that any script or user depended on the historical
90182 behavior.

90183 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**
90184 commands to be discarded. POSIX.1-2017 disallows this for consistency with mapped keys, the
90185 **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

90186 Input Editing in *ex*

90187 One of the common uses of the historical *ex* editor is over slow network connections. Editors that
 90188 run in canonical mode can require far less traffic to and from, and far less processing on, the host
 90189 machine, as well as more easily supporting block-mode terminals. For these reasons,
 90190 POSIX.1-2017 requires that *ex* be implemented using canonical mode input processing, as was
 90191 done historically.

90192 POSIX.1-2017 does not require the historical 4 BSD input editing characters “word erase” or
 90193 “literal next”. For this reason, it is unspecified how they are handled by *ex*, although they must
 90194 have the required effect. Implementations that resolve them after the line has been ended using a
 90195 <newline> or <control>-M character, and implementations that rely on the underlying system
 90196 terminal support for this processing, are both conforming. Implementations are strongly urged
 90197 to use the underlying system functionality, if at all possible, for compatibility with other system
 90198 text input interfaces.

90199 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor
 90200 moved to display the new end of the **autoindent** characters, but did not move the cursor to a
 90201 new line, nor did it erase the <control>-D character from the line. POSIX.1-2017 does not specify
 90202 that the cursor remain on the same line or that the rest of the line is erased; however,
 90203 implementations are strongly encouraged to provide the best possible user interface; that is, the
 90204 cursor should remain on the same line, and any <control>-D character on the line should be
 90205 erased.

90206 POSIX.1-2017 does not require the historical 4 BSD input editing character “reprint”,
 90207 traditionally <control>-R, which redisplayed the current input from the user. For this reason,
 90208 and because the functionality cannot be implemented after the line has been terminated by the
 90209 user, POSIX.1-2017 makes no requirements about this functionality. Implementations are
 90210 strongly urged to make this historical functionality available, if possible.

90211 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.
 90212 POSIX.1-2017 requires conformance to historical practice to avoid breaking historical *ex* scripts
 90213 and **.exrc** files.

90214 **eof**

90215 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left
 90216 unspecified so that implementations can conform in the presence of systems that do not support
 90217 this functionality. Implementations are encouraged to modify the line and redisplay it
 90218 immediately, if possible.

90219 The specification of the handling of the *eof* character differs from historical practice only in that
 90220 *eof* characters are not discarded if they follow normal characters in the text input. Historically,
 90221 they were always discarded.

90222 Command Descriptions in *ex*

90223 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and
 90224 **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit
 90225 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or
 90226 command execution in an empty file, make sense only for commands that add new text to the
 90227 edit buffer or write commands (because users may wish to write empty files). POSIX.1-2017
 90228 requires this behavior for such commands and disallows it otherwise, for consistency and
 90229 simplicity of specification.

90230 A count to an *ex* command has been historically corrected to be no greater than the last line in a

90231 file; for example, in a five-line file, the command **1,6print** would fail, but the command
90232 **1print300** would succeed. POSIX.1-2017 requires conformance to historical practice.

90233 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as
90234 described by POSIX.1-2017, but there were some special cases. For instance, the **list**, **number**,
90235 and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line
90236 3, and 3 would be the current line after the execution of the command. The **open** and **visual**
90237 commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the
90238 **open** and **visual** commands interacted badly with the **list** edit option, and setting and then
90239 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the
90240 specified format. For consistency and simplicity of specification, POSIX.1-2017 does not permit
90241 any of these exceptions to the general rule.

90242 POSIX.1-2017 uses the word *copy* in several places when discussing buffers. This is not intended
90243 to imply implementation.

90244 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would
90245 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a
90246 *count*. POSIX.1-2017 requires conformance to historical practice by default, but does not
90247 preclude extensions.

90248 Historically, the contents of the unnamed buffer were frequently discarded after commands that
90249 did not explicitly affect it; for example, when using the **edit** command to switch files. For
90250 consistency and simplicity of specification, POSIX.1-2017 does not permit this behavior.

90251 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting
90252 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user
90253 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric
90254 buffers would not have changed. POSIX.1-2017 requires conformance to historical practice.
90255 Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a
90256 single location in POSIX.1-2017.

90257 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as
90258 does the method for doing shell expansion. Implementations wishing to provide users with the
90259 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit
90260 option.

90261 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to
90262 do so; for example, **!date > /dev/null** does not require a screen refresh because the output of
90263 the UNIX *date* command requires only a single line of the screen. POSIX.1-2017 requires that the
90264 screen be refreshed if it has been overwritten, but makes no requirements as to how an
90265 implementation should make that determination. Implementations may prompt and refresh the
90266 screen regardless.

90267 Abbreviate

90268 Historical practice was that characters that were entered as part of an abbreviation replacement
90269 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,
90270 and so on; that is, they were logically pushed onto the terminal input queue, and were not a
90271 simple replacement. POSIX.1-2017 requires conformance to historical practice. Historical
90272 practice was that whenever a non-word character (that had not been escaped by a <control>-V)
90273 was entered after a word character, *vi* would check for abbreviations. The check was based on
90274 the type of the character entered before the word character of the word/non-word pair that
90275 triggered the check. The word character of the word/non-word pair that triggered the check and
90276 all characters entered before the trigger pair that were of that type were included in the check,
90277 with the exception of <blank> characters, which always delimited the abbreviation.

90278 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can
 90279 be no transitions from word to non-word characters (or *vice versa*) other than between the last
 90280 and next-to-last characters in the *lhs*, and there can be no <blank> characters in the *lhs*. In
 90281 addition, because of the historical quoting rules, it was impossible to enter a literal <control>-V
 90282 in the *lhs*. POSIX.1-2017 requires conformance to historical practice. Historical implementations
 90283 did not inform users when abbreviations that could never be used were entered;
 90284 implementations are strongly encouraged to do so.

90285 For example, the following abbreviations will work:

```
90286 :ab (p REPLACE
90287 :ab p REPLACE
90288 :ab ((p REPLACE
```

90289 The following abbreviations will not work:

```
90290 :ab ( REPLACE
90291 :ab (pp REPLACE
```

90292 Historical practice is that words on the *vi* colon command line were subject to abbreviation
 90293 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**
 90294 command. Because there are implementations that do not do abbreviation expansion for the first
 90295 argument to those commands, this is permitted, but not required, by POSIX.1-2017. However,
 90296 the following sequence:

```
90297 :ab foo bar
90298 :ab foo baz
```

90299 resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and
 90300 the sequence:

```
90301 :ab foo1 bar
90302 :ab foo2 bar
90303 :unabbreviate foo2
```

90304 deleted the abbreviation "foo1", not "foo2". These behaviors are not permitted by
 90305 POSIX.1-2017 because they clearly violate the expectations of the user.

90306 It was historical practice that <control>-V, not <backslash>, characters be interpreted as escaping
 90307 subsequent characters in the **abbreviate** command. POSIX.1-2017 requires conformance to
 90308 historical practice; however, it should be noted that an abbreviation containing a <blank> will
 90309 never work.

90310 Append

90311 Historically, any text following a <vertical-line> command separator after an **append**, **change**, or
 90312 **insert** command became part of the insert text. For example, in the command:

```
90313 :g/pattern/append|stuff1
```

90314 a line containing the text "stuff1" would be appended to each line matching pattern. It was
 90315 also historically valid to enter:

```
90316 :append|stuff1
90317 stuff2
90318 .
```

90319 and the text on the *ex* command line would be appended along with the text inserted after it.
 90320 There was an historical bug, however, that the user had to enter two terminating lines (the ' . '

90321 lines) to terminate text input mode in this case. POSIX.1-2017 requires conformance to historical
 90322 practice, but disallows the historical need for multiple terminating lines.

90323 **Change**

90324 See the RATIONALE for the **append** command. Historical practice for cursor positioning after
 90325 the change command when no text is input, is as described in POSIX.1-2017. However, one
 90326 System V implementation is known to have been modified such that the cursor is positioned on
 90327 the first address specified, and not on the line before the first address. POSIX.1-2017 disallows
 90328 this modification for consistency.

90329 Historically, the **change** command did not support buffer arguments, although some
 90330 implementations allow the specification of an optional buffer. This behavior is neither required
 90331 nor disallowed by POSIX.1-2017.

90332 **Change Directory**

90333 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as
 90334 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have
 90335 ' . ' or " . . " as their first component. Elements in the **cdpath** edit option are <colon>-separated.
 90336 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment
 90337 variable. This feature was not included in POSIX.1-2017 because it does not exist in any of the
 90338 implementations considered historical practice.

90339 **Copy**

90340 Historical implementations of *ex* permitted copies to lines inside of the specified range; for
 90341 example, **:2,5copy3** was a valid command. POSIX.1-2017 requires conformance to historical
 90342 practice.

90343 **Delete**

90344 POSIX.1-2017 requires support for the historical parsing of a **delete** command followed by flags,
 90345 without any intervening <blank> characters. For example:

90346 **1dp** Deletes the first line and prints the line that was second.

90347 **1delep** As for **1dp**.

90348 **1d** Deletes the first line, saving it in buffer *p*.

90349 **1d p1l** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was
 90350 second.

90351 **Edit**

90352 Historically, any *ex* command could be entered as a *+command* argument to the **edit** command,
 90353 although some (for example, **insert** and **append**) were known to confuse historical
 90354 implementations. For consistency and simplicity of specification, POSIX.1-2017 requires that any
 90355 command be supported as an argument to the **edit** command.

90356 Historically, the command argument was executed with the current line set to the last line of the
 90357 file, regardless of whether the **edit** command was executed from visual mode or not.
 90358 POSIX.1-2017 requires conformance to historical practice.

90359 Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first
 90360 <blank>, and there was no way to quote them. For consistency, POSIX.1-2017 requires that the
 90361 usual *ex* backslash quoting be provided.

90362 Historically, specifying the *+command* argument to the edit command required a filename to be
90363 specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of
90364 specification, POSIX.1-2017 does not permit this usage to fail for that reason.

90365 Historically, only the cursor position of the last file edited was remembered by the editor.
90366 POSIX.1-2017 requires that this be supported; however, implementations are permitted to
90367 remember and restore the cursor position for any file previously edited.

90368 **File**

90369 Historical versions of the *ex* editor **file** command displayed a current line and number of lines in
90370 the edit buffer of 0 when the file was empty, while the *vi* **<control>-G** command displayed a
90371 current line and number of lines in the edit buffer of 1 in the same situation. POSIX.1-2017 does
90372 not permit this discrepancy, instead requiring that a message be displayed indicating that the file
90373 is empty.

90374 **Global**

90375 The two-pass operation of the **global** and **v** commands is not intended to imply implementation,
90376 only the required result of the operation.

90377 The current line and column are set as specified for the individual *ex* commands. This
90378 requirement is cumulative; that is, the current line and column must track across all the
90379 commands executed by the **global** or **v** commands.

90380 **Insert**

90381 See the RATIONALE for the **append** command.

90382 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer
90383 was empty. POSIX.1-2017 requires that this command behave consistently with the **append**
90384 command.

90385 **Join**

90386 The action of the **join** command in relation to the special characters is only defined for the
90387 POSIX locale because the correct amount of white space after a period varies; in Japanese none is
90388 required, in French only a single space, and so on.

90389 **List**

90390 The historical output of the **list** command was potentially ambiguous. The standard developers
90391 believed correcting this to be more important than adhering to historical practice, and
90392 POSIX.1-2017 requires unambiguous output.

90393 **Map**

90394 Historically, command mode maps only applied to command names; for example, if the
90395 character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y'
90396 character. POSIX.1-2017 requires this behavior. Historically, entering **<control>-V** as the first
90397 character of a *vi* command was an error. Several implementations have extended the semantics
90398 of *vi* such that **<control>-V** means that the subsequent command character is not mapped. This
90399 is permitted, but not required, by POSIX.1-2017. Regardless, using **<control>-V** to escape the
90400 second or later character in a sequence of characters that might match a **map** command, or any
90401 character in text input mode, is historical practice, and stops the entered keys from matching a
90402 map. POSIX.1-2017 requires conformance to historical practice.

90403 Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored
90404 the map. POSIX.1-2017 requires that the mapped digits not be ignored.

90405 The historical implementation of the **map** command did not permit **map** commands that were
90406 more than a single character in length if the first character was printable. This behavior is
90407 permitted, but not required, by POSIX.1-2017.

90408 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the
90409 prefix of the mapped characters matched the mapping characters; for example, in the **map**:

```
90410 :map ab abcd
```

90411 the characters "ab" were used as is and were not remapped, but the characters "cd" were
90412 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.
90413 POSIX.1-2017 requires conformance to historical practice, and that such loops be interruptible.

90414 Text input maps had the same problems with expanding the *lhs* for the **ex map!** and **unmap!**
90415 command as did the **ex abbreviate** and **unabbreviate** commands. See the RATIONALE for the **ex**
90416 **abbreviate** command. POSIX.1-2017 requires similar modification of some historical practice for
90417 the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.

90418 Historically, **maps** that were subsets of other **maps** behaved differently depending on the order
90419 in which they were defined. For example:

```
90420 :map! ab      short  
90421 :map! abc     long
```

90422 would always translate the characters "ab" to "short", regardless of how fast the characters
90423 "abc" were entered. If the entry order was reversed:

```
90424 :map! abc     long  
90425 :map! ab      short
```

90426 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,
90427 and the characters might never be mapped to "short". For consistency and simplicity of
90428 specification, POSIX.1-2017 requires that the shortest match be used at all times.

90429 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified
90430 because the timing capabilities of systems are often inexact and variable, and it may depend on
90431 other factors such as the speed of the connection. The time should be long enough for the user to
90432 be able to complete the sequence, but not long enough for the user to have to wait. Some
90433 implementations of *vi* have added a **keytime** option, which permits users to set the number of
90434 0,1 seconds the editor waits for the completing characters. Because mapped terminal function
90435 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input
90436 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,
90437 or, at least timed out differently.

90438 **Mark**

90439 Historically, users were able to set the "previous context" marks explicitly. In addition, the **ex**
90440 commands " and ^ and the *vi* commands ", ` , ' , and " all referred to the same mark. In addition,
90441 the previous context marks were not set if the command, with which the address setting the
90442 mark was associated, failed. POSIX.1-2017 requires conformance to historical practice.
90443 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the
90444 change was undone. POSIX.1-2017 requires conformance to historical practice.

90445 The description of the special events that set the ` and ' marks matches historical practice. For
90446 example, historically the command **/a/,b/** did not set the ` and ' marks, but the command

90447 `/a/,b/delete` did.

90448 **Next**

90449 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,
90450 although some (for example, **insert** and **append**) were known to confuse historical
90451 implementations. POSIX.1-2017 requires that any command be permitted and that it behave as
90452 specified. The **next** command can accept more than one file, so usage such as:

```
90453 next `ls [abc] `
```

90454 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect
90455 only one filename.

90456 Historically, the **next** command behaved differently from the **:rewind** command in that it
90457 ignored the force flag if the **autowrite** flag was set. For consistency, POSIX.1-2017 does not
90458 permit this behavior.

90459 Historically, the **next** command positioned the cursor as if the file had never been edited before,
90460 regardless. POSIX.1-2017 does not permit this behavior, for consistency with the **edit** command.

90461 Implementations wanting to provide a counterpart to the **next** command that edited the
90462 previous file have used the command **prev[ious]**, which takes no *file* argument. POSIX.1-2017
90463 does not require this command.

90464 **Open**

90465 Historically, the **open** command would fail if the **open** edit option was not set. POSIX.1-2017
90466 does not mention the **open** edit option and does not require this behavior. Some historical
90467 implementations do not permit entering open mode from open or visual mode, only from *ex*
90468 mode. For consistency, POSIX.1-2017 does not permit this behavior.

90469 Historically, entering open mode from the command line (that is, *vi +open*) resulted in
90470 anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command
90471 `<control>-G` did not work. For consistency, POSIX.1-2017 does not permit this behavior.

90472 Historically, the **open** command only permitted ' / ' characters to be used as the search pattern
90473 delimiter. For consistency, POSIX.1-2017 requires that the search delimiters used by the **s**, **global**,
90474 and **v** commands be accepted as well.

90475 **Preserve**

90476 The **preserve** command does not historically cause the file to be considered unmodified for the
90477 purposes of future commands that may exit the editor. POSIX.1-2017 requires conformance to
90478 historical practice.

90479 Historical documentation stated that mail was not sent to the user when preserve was executed;
90480 however, historical implementations did send mail in this case. POSIX.1-2017 requires
90481 conformance to the historical implementations.

90482 **Print**

90483 The writing of NUL by the **print** command is not specified as a special case because the standard
90484 developers did not want to require *ex* to support NUL characters. Historically, characters were
90485 displayed using the ARPA standard mappings, which are as follows:

- 90486 1. Printable characters are left alone.
- 90487 2. Control characters less than \177 are represented as '^' followed by the character offset
90488 from the '@' character in the ASCII map; for example, \007 is represented as '^G'.
- 90489 3. \177 is represented as '^?' followed by '? '.

90490 The display of characters having their eighth bit set was less standard. Existing implementations
90491 use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their
90492 eighth bit set as the two characters "M-" followed by the seven-bit display as described above.)
90493 The latter probably has the best claim to historical practice because it was used for the **-v** option
90494 of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

90495 No specific display format is required by POSIX.1-2017.

90496 Explicit dependence on the ASCII character set has been avoided where possible, hence the use
90497 of the phrase an "implementation-defined multi-character sequence" for the display of non-
90498 printable characters in preference to the historical usage of, for instance, "^I" for the <tab>.
90499 Implementations are encouraged to conform to historical practice in the absence of any strong
90500 reason to diverge.

90501 Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized
90502 versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command
90503 names. POSIX.1-2017 permits, but does not require, this historical practice because capital forms
90504 of the commands are used by some implementations for other purposes.

90505 **Put**

90506 Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open
90507 or visual mode **P** command, if the buffer was named and was cut in character mode, and the
90508 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer
90509 was the source of the text, the entire line from which the text was taken was usually **put**, and the
90510 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.
90511 In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in
90512 errors as well, such as appending text that was unrelated to the (supposed) contents of the
90513 buffer. For consistency and simplicity of specification, POSIX.1-2017 does not permit these
90514 behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of the
90515 buffers are not altered by changing the mode of the editor.

90516 **Read**

90517 Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file,
90518 left an empty line as the first line of the file. For consistency and simplicity of specification,
90519 POSIX.1-2017 does not permit this behavior. Historically, a **read** in open or visual mode from a
90520 program left the cursor at the last line read in, not the first. For consistency, POSIX.1-2017 does
90521 not permit this behavior.

90522 Historical implementations of *ex* were unable to undo **read** commands that read from the output
90523 of a program. For consistency, POSIX.1-2017 does not permit this behavior.

90524 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified
90525 "characters", not "bytes". POSIX.1-2017 requires that the number of bytes be displayed, not the

90526 number of characters, because it may be difficult in multi-byte implementations to determine the
 90527 number of characters read. Implementations are encouraged to clarify the message displayed to
 90528 the user.

90529 Historically, reads were not permitted on files other than type regular, except that FIFO files
 90530 could be read (probably only because they did not exist when *ex* and *vi* were originally written).
 90531 Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way
 90532 to force the read. POSIX.1-2017 permits, but does not require, this behavior.

90533 **Recover**

90534 Some historical implementations of the editor permitted users to recover the edit buffer contents
 90535 from a previous edit session, and then exit without saving those contents (or explicitly
 90536 discarding them). The intent of POSIX.1-2017 in requiring that the edit buffer be treated as
 90537 already modified is to prevent this user error.

90538 **Rewind**

90539 Historical implementations supported the **rewind** command when the user was editing the first
 90540 file in the list; that is, the file that the **rewind** command would edit. POSIX.1-2017 requires
 90541 conformance to historical practice.

90542 **Substitute**

90543 Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the
 90544 last regular expression used in any command as the pattern, the same as the **~** command. The **r**
 90545 option is not required by POSIX.1-2017. Historically, the **c** and **g** options were toggled; for
 90546 example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of
 90547 specification, POSIX.1-2017 does not permit this behavior.

90548 The tilde command is often used to replace the last search RE. For example, in the sequence:

```
90549 s/red/blue/  
90550 /green  
90551 ~
```

90552 the **~** command is equivalent to:

```
90553 s/green/blue/
```

90554 Historically, *ex* accepted all of the following forms:

```
90555 s/abc/def/  
90556 s/abc/def  
90557 s/abc/  
90558 s/abc
```

90559 POSIX.1-2017 requires conformance to this historical practice.

90560 The **s** command presumes that the **^** character only occupies a single column in the display.
 90561 Much of the *ex* and *vi* specification presumes that the **<space>** only occupies a single column in
 90562 the display. There are no known character sets for which this is not true.

90563 Historically, the final column position for the substitute commands was based on previous
 90564 column movements; a search for a pattern followed by a substitution would leave the column
 90565 position unchanged, while a **0** command followed by a substitution would change the column
 90566 position to the first non-**<blank>**. For consistency and simplicity of specification, POSIX.1-2017
 90567 requires that the final column position always be set to the first non-**<blank>**.

90568

Set

90569

90570

Historical implementations redisplayed all of the options for each occurrence of the **all** keyword. POSIX.1-2017 permits, but does not require, this behavior.

90571

Tag

90572

90573

90574

90575

90576

90577

No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry. Historical practice has been to look for the path found in the **tags** file, based on the current directory. A useful extension found in some implementations is to look based on the directory containing the tags file that held the entry, as well. No requirement is made as to which reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such as multiple entries in a tags file for a tag.

90578

90579

90580

90581

Because users often specify many different tags files, some of which need not be relevant or exist at any particular time, POSIX.1-2017 requires that error messages about problem tags files be displayed only if the requested tag is not found, and then, only once for each time that the **tag** edit option is changed.

90582

90583

90584

90585

90586

90587

The requirement that the current edit buffer be unmodified is only necessary if the file indicated by the tag entry is not the same as the current file (as defined by the current pathname). Historically, the file would be reloaded if the filename had changed, as well as if the filename was different from the current pathname. For consistency and simplicity of specification, POSIX.1-2017 does not permit this behavior, requiring that the name be the only factor in the decision.

90588

90589

90590

90591

Historically, *vi* only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor were not found. POSIX.1-2017 considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

90592

Undo

90593

90594

90595

The **undo** description deliberately uses the word “modified”. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

90596

90597

90598

90599

90600

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. POSIX.1-2017 requires a simplified behavior for consistency and simplicity of specification.

90601

Version

90602

90603

90604

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

90605 Write

90606 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified
90607 “characters”, not “bytes”. POSIX.1-2017 requires that the number of bytes be displayed, not the
90608 number of characters because it may be difficult in multi-byte implementations to determine the
90609 number of characters written. Implementations are encouraged to clarify the message displayed
90610 to the user.

90611 Implementation-defined tests are permitted so that implementations can make additional
90612 checks; for example, for locks or file modification times.

90613 Historically, attempting to append to a nonexistent file caused an error. It has been left
90614 unspecified in POSIX.1-2017 to permit implementations to let the **write** succeed, so that the
90615 append semantics are similar to those of the historical *cs*.

90616 Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around
90617 dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote
90618 them as files of a single, empty line. POSIX.1-2017 does not permit this behavior.

90619 Historically, *ex* restored standard output and standard error to their values as of when *ex* was
90620 invoked, before writes to programs were performed. This could disturb the terminal
90621 configuration as well as be a security issue for some terminals. POSIX.1-2017 does not permit
90622 this, requiring that the program output be captured and displayed as if by the *ex* **print**
90623 command.

90624 Adjust Window

90625 Historically, the line count was set to the value of the **scroll** option if the type character was end-
90626 of-file. This feature was broken on most historical implementations long ago, however, and is
90627 not documented anywhere. For this reason, POSIX.1-2017 is resolutely silent.

90628 Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+**
90629 and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =**
90630 were historically invalid.) POSIX.1-2017 requires conformance to this historical practice.

90631 Historically, the **z** command was further <blank>-sensitive in that the *count* could not be
90632 <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the
90633 *count* is not ambiguous with respect to either the type character or the flags, this is not permitted
90634 by POSIX.1-2017.

90635 Escape

90636 Historically, *ex* filter commands only read the standard output of the commands, letting
90637 standard error appear on the terminal as usual. The *vi* utility, however, read both standard
90638 output and standard error. POSIX.1-2017 requires the latter behavior for both *ex* and *vi*, for
90639 consistency.

90640 Shift Left and Shift Right

90641 Historically, it was possible to add shift characters to increase the effect of the command; for
90642 example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default
90643 1. POSIX.1-2017 requires conformance to historical practice.

90644 <control>-D

90645 Historically, the <control>-D command erased the prompt, providing the user with an unbroken
 90646 presentation of lines from the edit buffer. This is not required by POSIX.1-2017; implementations
 90647 are encouraged to provide it if possible. Historically, the <control>-D command took, and then
 90648 ignored, a *count*. POSIX.1-2017 does not permit this behavior.

90649 Write Line Number

90650 Historically, the *ex =* command, when executed in *ex* mode in an empty edit buffer, reported 0,
 90651 and from open or visual mode, reported 1. For consistency and simplicity of specification,
 90652 POSIX.1-2017 does not permit this behavior.

90653 Execute

90654 Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**,
 90655 **insert**, and **change**) in executed buffers. POSIX.1-2017 does not permit this exclusion for
 90656 consistency.

90657 Historically, the logical contents of the buffer being executed did not change if the buffer itself
 90658 were modified by the commands being executed; that is, buffer execution did not support self-
 90659 modifying code. POSIX.1-2017 requires conformance to historical practice.

90660 Historically, the @ command took a range of lines, and the @ buffer was executed once per line,
 90661 with the current line ('.') set to each specified line. POSIX.1-2017 requires conformance to
 90662 historical practice.

90663 Some historical implementations did not notice if errors occurred during buffer execution. This,
 90664 coupled with the ability to specify a range of lines for the *ex @* command, makes it trivial to
 90665 cause them to drop **core**. POSIX.1-2017 requires that implementations stop buffer execution if
 90666 any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are
 90667 replaced (for example, the buffer executes the *ex :edit* command).

90668 Regular Expressions in *ex*

90669 Historical practice is that the characters in the replacement part of the last *s* command ~~that~~ that is,
 90670 those matched by entering a '~' in the regular expression ~~that were~~ not further expanded by the
 90671 regular expression engine. So, if the characters contained the string "a.", they would match
 90672 'a' followed by ".", and not 'a' followed by any character. POSIX.1-2017 requires
 90673 conformance to historical practice.

90674 Edit Options in *ex*

90675 The following paragraphs describe the historical behavior of some edit options that were not, for
 90676 whatever reason, included in POSIX.1-2017. Implementations are strongly encouraged to only
 90677 use these names if the functionality described here is fully supported.

90678 **extended** The **extended** edit option has been used in some implementations of *vi* to provide
 90679 extended regular expressions instead of basic regular expressions. This option was
 90680 omitted from POSIX.1-2017 because it is not widespread historical practice.

90681 **flash** The **flash** edit option historically caused the screen to flash instead of beeping on
 90682 error. This option was omitted from POSIX.1-2017 because it is not found in some
 90683 historical implementations.

90684	hardtabs	The hardtabs edit option historically defined the number of columns between hardware tab settings. This option was omitted from POSIX.1-2017 because it was believed to no longer be generally useful.
90685		
90686		
90687	modeline	The modeline (sometimes named modelines) edit option historically caused <i>ex</i> or <i>vi</i> to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.
90688		
90689		
90690		
90691	open	The open edit option historically disallowed the <i>ex</i> open and visual commands. This edit option was omitted because these commands are required by POSIX.1-2017.
90692		
90693		
90694	optimize	The optimize edit option historically expedited text throughput by setting the terminal to not do automatic <carriage-return> characters when printing more than one logical line of output. This option was omitted from POSIX.1-2017 because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.
90695		
90696		
90697		
90698		
90699	ruler	The ruler edit option has been used in some implementations of <i>vi</i> to present a current row/column ruler for the user. This option was omitted from POSIX.1-2017 because it is not widespread historical practice.
90700		
90701		
90702	sourceany	The sourceany edit option historically caused <i>ex</i> or <i>vi</i> to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.
90703		
90704		
90705		
90706	timeout	The timeout edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from POSIX.1-2017 because its behavior is now standard, it is not widely useful, and it was rarely documented.
90707		
90708		
90709		
90710	verbose	The verbose edit option has been used in some implementations of <i>vi</i> to cause <i>vi</i> to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical <i>vi</i> only alerted the terminal and presented no message for such errors. The historical editor option terse did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from POSIX.1-2017 because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.
90711		
90712		
90713		
90714		
90715		
90716		
90717		
90718		
90719	wraplen	The wraplen edit option has been used in some implementations of <i>vi</i> to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This option was omitted from POSIX.1-2017 because it is not widespread historical practice; however, implementors are encouraged to use it if they add this functionality.
90720		
90721		
90722		
90723		
90724		

90725 **autoindent, ai**

90726 Historically, the command **0a** did not do any autoindentation, regardless of the current
90727 indentation of line 1. POSIX.1-2017 requires that any indentation present in line 1 be used.

90728 **autoprint, ap**

90729 Historically, the **autoprint** edit option was not completely consistent or based solely on
90730 modifications to the edit buffer. Exceptions were the **read** command (when reading from a file,
90731 but not from a filter), the **append**, **change**, **insert**, **global**, and **v** commands, all of which were not
90732 affected by **autoprint**, and the **tag** command, which was affected by **autoprint**. POSIX.1-2017
90733 requires conformance to historical practice.

90734 Historically, the **autoprint** option only applied to the last of multiple commands entered using
90735 <vertical-line> delimiters; for example, **delete** <newline> was affected by **autoprint**, but
90736 **delete | version** <newline> was not. POSIX.1-2017 requires conformance to historical practice.

90737 **autowrite, aw**

90738 Appending the '!' character to the *ex* **next** command to avoid performing an automatic write
90739 was not supported in historical implementations. POSIX.1-2017 requires that the behavior match
90740 the other *ex* commands for consistency.

90741 **ignorecase, ic**

90742 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to
90743 counter-intuitive situations when uppercase characters were used in range expressions.
90744 Historically, the process was as follows:

- 90745 1. Take a line of text from the edit buffer.
- 90746 2. Convert uppercase to lowercase in text line.
- 90747 3. Convert uppercase to lowercase in regular expressions, except in character class
90748 specifications.
- 90749 4. Match regular expressions against text.

90750 This would mean that, with **ignorecase** in effect, the text:

90751 `The cat sat on the mat`

90752 would be matched by

90753 `/^the/`

90754 but not by:

90755 `/^[A-Z]he/`

90756 For consistency with other commands implementing regular expressions, POSIX.1-2017 does not
90757 permit this behavior.

90758 **paragraphs, para**

90759 The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options
 90760 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text
 90761 formatter, and a “portable user” could use the {, }, [[,]], (, and) commands in open or visual
 90762 mode and have the cursor stop in unexpected places. POSIX.1-2017 specifies their values in the
 90763 POSIX locale because the unusual grouping (they only work when grouped into two characters
 90764 at a time) means that they cannot be used for general-purpose movement, regardless.

90765 **readonly**

90766 Implementations are encouraged to provide the best possible information to the user as to the
 90767 read-only status of the file, with the exception that they should not consider the current special
 90768 privileges of the process. This provides users with a safety net because they must force the
 90769 overwrite of read-only files, even when running with additional privileges.

90770 The **readonly** edit option specification largely conforms to historical practice. The only
 90771 difference is that historical implementations did not notice that the user had set the **readonly**
 90772 edit option in cases where the file was already marked read-only for some reason, and would
 90773 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were
 90774 replaced. This behavior is disallowed by POSIX.1-2017.

90775 **report**

90776 The requirement that lines copied to a buffer interact differently than deleted lines is historical
 90777 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be
 90778 written, but 4 lines must be copied before a report is written.

90779 The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based
 90780 on the total number of lines added or deleted during the command execution, and that
 90781 commands executed by the **global** and **v** commands not present reports, is historical practice.
 90782 POSIX.1-2017 extends historical practice by requiring that buffer execution be treated similarly.
 90783 The reasons for this are two-fold. Historically, only the report by the last command executed
 90784 from the buffer would be seen by the user, as each new report would overwrite the last. In
 90785 addition, the standard developers believed that buffer execution had more in common with
 90786 **global** and **v** commands than it did with other *ex* commands, and should behave similarly, for
 90787 consistency and simplicity of specification.

90788 **showmatch, sm**

90789 The length of time the cursor spends on the matching character is unspecified because the
 90790 timing capabilities of systems are often inexact and variable. The time should be long enough for
 90791 the user to notice, but not long enough for the user to become annoyed. Some implementations
 90792 of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals
 90793 the cursor pauses on the matching character.

90794 **showmode**

90795 The **showmode** option has been used in some historical implementations of *ex* and *vi* to display
 90796 the current editing mode when in open or visual mode. The editing modes have generally
 90797 included “command” and “input”, and sometimes other modes such as “replace” and
 90798 “change”. The string was usually displayed on the bottom line of the screen at the far right-hand
 90799 corner. In addition, a preceding '*' character often denoted whether the contents of the edit
 90800 buffer had been modified. The latter display has sometimes been part of the **showmode** option,
 90801 and sometimes based on another option. This option was not available in the 4 BSD historical

90802 implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is
90803 required by POSIX.1-2017.

90804 The **smd** shorthand for the **showmode** option was not present in all historical implementations
90805 of the editor. POSIX.1-2017 requires it, for consistency.

90806 Not all historical implementations of the editor displayed a mode string for command mode,
90807 differentiating command mode from text input mode by the absence of a mode string.
90808 POSIX.1-2017 permits this behavior for consistency with historical practice, but implementations
90809 are encouraged to provide a display string for both modes.

90810 **slowopen**

90811 Historically, the **slowopen** option was automatically set if the terminal baud rate was less than
90812 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen**
90813 option had two effects. First, when inserting characters in the middle of a line, characters after
90814 the cursor would not be pushed ahead, but would appear to be overwritten. Second, when
90815 creating a new line of text, lines after the current line would not be scrolled down, but would
90816 appear to be overwritten. In both cases, ending text input mode would cause the screen to be
90817 refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently
90818 intelligent caused the editor to ignore the **slowopen** option. POSIX.1-2017 permits most
90819 historical behavior, extending historical practice to require **slowopen** behaviors if the edit option
90820 is set by the user.

90821 **tags**

90822 The default path for tags files is left unspecified as implementations may have their own **tags**
90823 implementations that do not correspond to the historical ones. The default **tags** option value
90824 should probably at least include the file **./tags**.

90825 **term**

90826 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial
90827 terminal information was loaded. This is permitted by POSIX.1-2017; however, implementations
90828 are encouraged to permit the user to modify their terminal type at any time.

90829 **terse**

90830 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,
90831 for some error messages. This is permitted, but not required, by POSIX.1-2017. Historically, most
90832 common visual mode errors (for example, trying to move the cursor past the end of a line) did
90833 not result in an error message, but simply alerted the terminal. Implementations wishing to
90834 provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not
90835 **terse**.

90836 **window**

90837 In historical implementations, the default for the **window** edit option was based on the baud
90838 rate as follows:

- 90839 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for
90840 example, the line:

```
90841 set w300=12
```

90842 would set the window option to 12 if the baud rate was less than 1 200.

- 90843 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.
- 90844 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

90845 The **w300**, **w1200**, and **w9600** options do not appear in POSIX.1-2017 because of their
90846 dependence on specific baud rates.

90847 In historical implementations, the size of the window displayed by various commands was
90848 related to, but not necessarily the same as, the **window** edit option. For example, the size of the
90849 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**
90850 edit option. However, changing the value of the **window** edit option did change the number of
90851 lines that were displayed when the screen was repainted. POSIX.1-2017 does not permit this
90852 behavior in the interests of consistency and simplicity of specification, and requires that all
90853 commands that change the number of lines that are displayed do it by setting the value of the
90854 **window** edit option.

90855 **wrapmargin, wm**

90856 Historically, the **wrapmargin** option did not affect maps inserting characters that also had
90857 associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used
90858 maps that depend on this behavior. For consistency and simplicity of specification,
90859 POSIX.1-2017 does not permit this behavior.

90860 Historically, **wrapmargin** was calculated using the column display width of all characters on the
90861 screen. For example, an implementation using "**^I**" to represent <tab> characters when the **list**
90862 edit option was set, where '**^**' and '**I**' each took up a single column on the screen, would
90863 calculate the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option
90864 similarly changed the effective length of the line as well. POSIX.1-2017 requires conformance to
90865 historical practice.

90866 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
90867 but this has been modified in this version.

90868 **FUTURE DIRECTIONS**

90869 None.

90870 **SEE ALSO**

90871 [Section 2.9.1.1](#) (on page 2367), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*

90872 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on
90873 page 216)

90874 XSH [access\(\)](#)

90875 **CHANGE HISTORY**

90876 First released in Issue 2.

90877 **Issue 5**

90878 The FUTURE DIRECTIONS section is added.

90879 **Issue 6**

90880 This utility is marked as part of the User Portability Utilities option.

90881 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

90882 The following new requirements on POSIX implementations derive from alignment with the
90883 Single UNIX Specification:

90884 In the **map** command description, the sequence *#digit* is added.

90885 The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

90886 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This
90887 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,
90888 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

90889 The **-l** option is removed.

90890 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.

90891 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/8 is applied, making an editorial
90892 correction in the EXTENDED DESCRIPTION.

90893 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/9 is applied, removing text describing
90894 behavior on systems with bytes consisting of more than eight bits.

90895 **Issue 7**

90896 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is
90897 ' _ '.

90898 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

90899 Austin Group Interpretation 1003.1-2001 #121 is applied, clarifying the *ex write* command.

90900 Austin Group Interpretation 1003.1-2001 #156 is applied.

90901 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

90902 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0093 [584] is applied.

90903 **NAME**90904 `expand` `‡`convert tabs to spaces90905 **SYNOPSIS**90906 `expand [-t tablist] [file...]`90907 **DESCRIPTION**

90908 The *expand* utility shall write files or the standard input to the standard output with <tab>
 90909 characters replaced with one or more <space> characters needed to pad to the next tab stop. Any
 90910 <backspace> characters shall be copied to the output and cause the column position count for
 90911 tab stop calculations to be decremented; the column position count shall not be decremented
 90912 below zero.

90913 **OPTIONS**90914 The *expand* utility shall conform to XBD [Section 12.2](#) (on page 216).

90915 The following option shall be supported:

90916 `-t tablist` Specify the tab stops. The application shall ensure that the argument *tablist* consists
 90917 of either a single positive decimal integer or a list of tabstops. If a single number is
 90918 given, tabs shall be set that number of column positions apart instead of the
 90919 default 8.

90920 If a list of tabstops is given, the application shall ensure that it consists of a list of
 90921 two or more positive decimal integers, separated by <blank> or <comma>
 90922 characters, in ascending order. The <tab> characters shall be set at those specific
 90923 column positions. Each tab stop *N* shall be an integer value greater than zero, and
 90924 the list is in strictly ascending order. This is taken to mean that, from the start of a
 90925 line of output, tabbing to position *N* shall cause the next character output to be in
 90926 the (*N*+1)th column position on that line.

90927 In the event of *expand* having to process a <tab> at a position beyond the last of
 90928 those specified in a multiple tab-stop list, the <tab> shall be replaced by a single
 90929 <space> in the output.

90930 **OPERANDS**

90931 The following operand shall be supported:

90932 *file* The pathname of a text file to be used as input.90933 **STDIN**

90934 See the INPUT FILES section.

90935 **INPUT FILES**

90936 Input files shall be text files.

90937 **ENVIRONMENT VARIABLES**90938 The following environment variables shall affect the execution of *expand*:

90939 *LANG* Provide a default value for the internationalization variables that are unset or null.
 90940 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 90941 variables used to determine the values of locale categories.)

90942 *LC_ALL* If set to a non-empty string value, override the values of all the other
 90943 internationalization variables.

90944 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 90945 characters (for example, single-byte as opposed to multi-byte characters in
 90946 arguments and input files), the processing of <tab> and <space> characters, and
 90947 for the determination of the width in column positions each character would

90948 occupy on an output device.

90949 *LC_MESSAGES*

90950 Determine the locale that should be used to affect the format and contents of

90951 diagnostic messages written to standard error.

90952 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

90953 **ASYNCHRONOUS EVENTS**

90954 Default.

90955 **STDOUT**

90956 The standard output shall be equivalent to the input files with <tab> characters converted into

90957 the appropriate number of <space> characters.

90958 **STDERR**

90959 The standard error shall be used only for diagnostic messages.

90960 **OUTPUT FILES**

90961 None.

90962 **EXTENDED DESCRIPTION**

90963 None.

90964 **EXIT STATUS**

90965 The following exit values shall be returned:

90966 0 Successful completion

90967 >0 An error occurred.

90968 **CONSEQUENCES OF ERRORS**

90969 The *expand* utility shall terminate with an error message and non-zero exit status upon

90970 encountering difficulties accessing one of the *file* operands.

90971 **APPLICATION USAGE**

90972 None.

90973 **EXAMPLES**

90974 None.

90975 **RATIONALE**

90976 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific

90977 columns, and so on) that contain <tab> characters.

90978 See XBD [Section 3.103](#) (on page 50).

90979 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8

90980 mandates that *expand* shall accept the integers (within the single argument) separated using

90981 either <comma> or <blank> characters.

90982 Earlier versions of this standard allowed the following form in the SYNOPSIS:

90983 `expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]`

90984 This form is no longer specified by POSIX.1-2017 but may be present in some implementations.

90985 **FUTURE DIRECTIONS**

90986 None.

90987 **SEE ALSO**90988 *tabs, unexpand*90989 XBD [Section 3.103](#) (on page 50), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)90990 **CHANGE HISTORY**

90991 First released in Issue 4.

90992 **Issue 6**

90993 This utility is marked as part of the User Portability Utilities option.

90994 The APPLICATION USAGE section is added.

90995 The obsolescent SYNOPSIS is removed.

90996 The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b draft standard.

90998 The normative text is reworded to avoid use of the term “must” for application requirements.

90999 **Issue 7**

91000 Austin Group Interpretation 1003.1-2001 #027 is applied.

91001 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91002 The *expand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

91003

91004 **NAME**91005 *expr* — evaluate arguments as an expression91006 **SYNOPSIS**91007 *expr operand...*91008 **DESCRIPTION**91009 The *expr* utility shall evaluate an expression and write the result to standard output.91010 **OPTIONS**

91011 None.

91012 **OPERANDS**91013 The single expression evaluated by *expr* shall be formed from the *operand* operands, as described in the EXTENDED DESCRIPTION section. The application shall ensure that each of the expression operator symbols:

91016 () | & = > >= < <= != + - * / % :

91017 and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.91018 **STDIN**

91019 Not used.

91020 **INPUT FILES**

91021 None.

91022 **ENVIRONMENT VARIABLES**91023 The following environment variables shall affect the execution of *expr*:91024 *LANG* Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)91027 *LC_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.91029 *LC_COLLATE*

91030 Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and by the string comparison operators.

91033 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes within regular expressions.91036 *LC_MESSAGES*

91037 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

91039 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.91040 **ASYNCHRONOUS EVENTS**

91041 Default.

91042 **STDOUT**91043 The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to standard output.

91045 **STDERR**

91046 The standard error shall be used only for diagnostic messages.

91047 **OUTPUT FILES**

91048 None.

91049 **EXTENDED DESCRIPTION**

91050 The formation of the expression to be evaluated is shown in the following table. The symbols
 91051 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the
 91052 expression operator symbols (all separate arguments) by recursive application of the constructs
 91053 described in the table. The expressions are listed in order of decreasing precedence, with equal-
 91054 precedence operators grouped between horizontal lines. All of the operators shall be left-
 91055 associative.

Expression	Description
<i>integer</i>	An argument consisting only of an (optional) unary minus followed by digits.
<i>string</i>	A string argument; see below.
(<i>expr</i>)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
<i>expr1</i> : <i>expr2</i>	Matching expression; see below.
<i>expr1</i> * <i>expr2</i> <i>expr1</i> / <i>expr2</i>	Multiplication of decimal integer-valued arguments. Integer division of decimal integer-valued arguments, producing an integer result.
<i>expr1</i> % <i>expr2</i>	Remainder of integer division of decimal integer-valued arguments.
<i>expr1</i> + <i>expr2</i> <i>expr1</i> - <i>expr2</i>	Addition of decimal integer-valued arguments. Subtraction of decimal integer-valued arguments.
<i>expr1</i> = <i>expr2</i> <i>expr1</i> > <i>expr2</i> <i>expr1</i> >= <i>expr2</i> <i>expr1</i> < <i>expr2</i> <i>expr1</i> <= <i>expr2</i> <i>expr1</i> != <i>expr2</i>	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false. Equal. Greater than. Greater than or equal. Less than. Less than or equal. Not equal.
<i>expr1</i> & <i>expr2</i>	Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.
<i>expr1</i> <i>expr2</i>	Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.

91087 **Matching Expression**

91088 The ':' matching operator shall compare the string resulting from the evaluation of *expr1* with
 91089 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax
 91090 shall be that defined in XBD [Section 9.3](#) (on page 183), except that all patterns are anchored to
 91091 the beginning of the string (that is, only sequences starting at the first character of a string are
 91092 matched by the regular expression) and, therefore, it is unspecified whether '^' is a special
 91093 character in that context. Usually, the matching operator shall return a string representing the
 91094 number of characters matched ('0' on failure). Alternatively, if the pattern contains at least one
 91095 regular expression subexpression "[\\(\\.\\.\\.\\)]", the string matched by the back-reference
 91096 expression "\\1" shall be returned. If the back-reference expression "\\1" does not match, then
 91097 the null string shall be returned.

91098 **Identification as Integer or String**

91099 An argument or the value of a subexpression that consists only of an optional unary minus
 91100 followed by digits is a candidate for treatment as an integer if it is used as the left argument to
 91101 the | operator or as either argument to any of the following operators: & = > >= < <= != +
 91102 - * / %. Otherwise, the argument or subexpression value shall be treated as a string.

91103 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

91104 **EXIT STATUS**

91105 The following exit values shall be returned:

- 91106 0 The *expression* evaluates to neither null nor zero.
- 91107 1 The *expression* evaluates to null or zero.
- 91108 2 Invalid *expression*.
- 91109 >2 An error occurred.

91110 **CONSEQUENCES OF ERRORS**

91111 Default.

91112 **APPLICATION USAGE**

91113 The *expr* utility has a rather difficult syntax:

91114 Many of the operators are also shell control operators or reserved words, so they have to
 91115 be escaped on the command line.

91116 Each part of the expression is composed of separate arguments, so liberal usage of <blank>
 91117 characters is required. For example:

91118

91119

91120

91121

91122

Invalid	Valid
<i>expr</i> 1+2	<i>expr</i> 1 + 2
<i>expr</i> "1 + 2"	<i>expr</i> 1 + 2
<i>expr</i> 1 + (2 * 3)	<i>expr</i> 1 + \(2 * 3 \)

91123

91124

91125

91126

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in *expr*. Newly written scripts should avoid *expr* in favor of the new features within the shell; see [Section 2.5](#) (on page 2349) and [Section 2.6.4](#) (on page 2358).

91127

91128

After argument processing by the shell, *expr* is not required to be able to tell the difference between an operator and an operand except by the value. If "\$a" is '=', the command:

91129

```
expr "$a" = '='
```

91130 looks like:

91131 `expr = = =`

91132 as the arguments are passed to *expr* (and they all may be taken as the '=' operator). The
91133 following works reliably:

91134 `expr "X$a" = X=`

91135 Also note that this volume of POSIX.1-2017 permits implementations to extend utilities. The *expr*
91136 utility permits the integer arguments to be preceded with a unary minus. This means that an
91137 integer argument could look like an option. Therefore, the conforming application must employ
91138 the "--" construct of Guideline 10 of XBD [Section 12.2](#) (on page 216) to protect its operands if
91139 there is any chance the first operand might be a negative integer (or any string with a leading
91140 minus).

91141 For testing string equality the *test* utility is preferred over *expr*, as it is usually implemented as a
91142 shell built-in. However, the functionality is not quite the same because the *expr =* and *!=*
91143 operators check whether strings collate equally, whereas *test* checks whether they are identical.
91144 Therefore, they can produce different results in locales where the collation sequence does not
91145 have a total ordering of all characters (see XBD [Section 7.3.2](#), on page 147).

91146 EXAMPLES

91147 The following command:

91148 `a=$(expr "$a" + 1)`

91149 adds 1 to the variable *a*.

91150 The following command, for "\$a" equal to either */usr/abc/file* or just *file*:

91151 `expr $a : '.*\/(.*\)' \| $a`

91152 returns the last segment of a pathname (that is, *file*). Applications should avoid the character
91153 '/' used alone as an argument; *expr* may interpret it as the division operator.

91154 The following command:

91155 `expr "//$a" : '.*\/(.*\)'`

91156 is a better representation of the previous example. The addition of the "//" characters
91157 eliminates any ambiguity about the division operator and simplifies the whole expression. Also
91158 note that pathnames may contain characters contained in the *IFS* variable and should be quoted
91159 to avoid having "\$a" expand into multiple arguments.

91160 The following command:

91161 `expr "X$VAR" : '.*' - 1`

91162 returns the number of characters in *VAR*.

91163 RATIONALE

91164 In an early proposal, EREs were used in the matching expression syntax. This was changed to
91165 BREs to avoid breaking historical applications.

91166 The use of a leading <circumflex> in the BRE is unspecified because many historical
91167 implementations have treated it as a special character, despite their system documentation. For
91168 example:

91169 `expr foo : ^foo` `expr ^foo : ^foo`

91170 return 3 and 0, respectively, on those systems; their documentation would imply the reverse.

- 91171 Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on
91172 this undocumented feature.
- 91173 **FUTURE DIRECTIONS**
91174 None.
- 91175 **SEE ALSO**
91176 [Section 2.5](#) (on page 2349), [Section 2.6.4](#) (on page 2358)
91177 XBD [Section 7.3.2](#) (on page 147), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#)
91178 (on page 216)
- 91179 **CHANGE HISTORY**
91180 First released in Issue 2.
- 91181 **Issue 5**
91182 The FUTURE DIRECTIONS section is added.
- 91183 **Issue 6**
91184 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE
91185 PASC Interpretation 1003.2 #104.
91186 The normative text is reworded to avoid use of the term “must” for application requirements.
- 91187 **Issue 7**
91188 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.
91189 The SYNOPSIS and OPERANDS sections are revised to explicitly state that the name of each of
91190 the operands is *operand*.
91191 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0094 [942], XCU/TC2-2008/0095
91192 [709], XCU/TC2-2008/0096 [942], XCU/TC2-2008/0097 [963], and XCU/TC2-2008/0098 [942]
91193 are applied.

91194 **NAME**
91195 false — return false value

91196 **SYNOPSIS**
91197 false

91198 **DESCRIPTION**
91199 The *false* utility shall return with a non-zero exit code.

91200 **OPTIONS**
91201 None.

91202 **OPERANDS**
91203 None.

91204 **STDIN**
91205 Not used.

91206 **INPUT FILES**
91207 None.

91208 **ENVIRONMENT VARIABLES**
91209 None.

91210 **ASYNCHRONOUS EVENTS**
91211 Default.

91212 **STDOUT**
91213 Not used.

91214 **STDERR**
91215 Not used.

91216 **OUTPUT FILES**
91217 None.

91218 **EXTENDED DESCRIPTION**
91219 None.

91220 **EXIT STATUS**
91221 The *false* utility shall always exit with a value other than zero.

91222 **CONSEQUENCES OF ERRORS**
91223 Default.

91224 **APPLICATION USAGE**
91225 None.

91226 **EXAMPLES**
91227 None.

91228 **RATIONALE**
91229 None.

91230 **FUTURE DIRECTIONS**
91231 None.

91232 **SEE ALSO**
91233 *true*

91234 **CHANGE HISTORY**

91235 First released in Issue 2.

91236 **Issue 6**

91237 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR
91238 section from ``None.'' to ``Not used.'' for alignment with [Section 1.4](#) (on page 2336).

91239 **NAME**91240 `fc` — process the command history list91241 **SYNOPSIS**

```
91242 UP fc [-r] [-e editor] [first [last]]
91243 fc -l [-nr] [first [last]]
91244 fc -s [old=new] [first]
```

91245 **DESCRIPTION**91246 The `fc` utility shall list, or shall edit and re-execute, commands previously entered to an
91247 interactive `sh`.

91248 The command history list shall reference commands by number. The first number in the list is
91249 selected arbitrarily. The relationship of a number to its command shall not change except when
91250 the user logs in and no other process is accessing the list, at which time the system may reset the
91251 numbering to start the oldest retained command at another number (usually 1). When the
91252 number reaches an implementation-defined upper limit, which shall be no smaller than the
91253 value in `HISTSIZE` or 32767 (whichever is greater), the shell may wrap the numbers, starting the
91254 next command with a lower number (usually 1). However, despite this optional wrapping of
91255 numbers, `fc` shall maintain the time-ordering sequence of the commands. For example, if four
91256 commands in sequence are given the numbers 32766, 32767, 1 (wrapped), and 2 as they are
91257 executed, command 32767 is considered the command previous to 1, even though its number is
91258 higher.

91259 When commands are edited (when the `-l` option is not specified), the resulting lines shall be
91260 entered at the end of the history list and then re-executed by `sh`. The `fc` command that caused the
91261 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this
91262 shall suppress the entry into the history list and the command re-execution. Any command line
91263 variable assignments or redirection operators used with `fc` shall affect both the `fc` command itself
91264 as well as the command that results; for example:

91265

```
fc -s -- -l 2>/dev/null
```

91266 reinvoke the previous command, suppressing standard error for both `fc` and the previous
91267 command.91268 **OPTIONS**91269 The `fc` utility shall conform to XBD [Section 12.2](#) (on page 216).

91270 The following options shall be supported:

- 91271 **-e editor** Use the editor named by *editor* to edit the commands. The *editor* string is a utility
91272 name, subject to search via the `PATH` variable (see XBD [Chapter 8](#), on page 173).
91273 The value in the `FCEDIT` variable shall be used as a default when `-e` is not
91274 specified. If `FCEDIT` is null or unset, `ed` shall be used as the editor.
- 91275 **-l** (The letter ell.) List the commands rather than invoking an editor on them. The
91276 commands shall be written in the sequence indicated by the *first* and *last* operands,
91277 as affected by `-r`, with each command preceded by the command number.
- 91278 **-n** Suppress command numbers when listing with `-l`.
- 91279 **-r** Reverse the order of the commands listed (with `-l`) or edited (with neither `-l` nor
91280 `-s`).

- 91281 -s Re-execute the command without invoking an editor.
- 91282 **OPERANDS**
- 91283 The following operands shall be supported:
- 91284 *first, last* Select the commands to list or edit. The number of previous commands that can be
91285 accessed shall be determined by the value of the *HISTSIZE* variable. The value of
91286 *first* or *last* or both shall be one of the following:
- 91287 [+]*number* A positive number representing a command number; command
91288 numbers can be displayed with the -l option.
- 91289 -*number* A negative decimal number representing the command that was
91290 executed *number* of commands previously. For example, -1 is the
91291 immediately previous command.
- 91292 *string* A string indicating the most recently entered command that begins
91293 with that string. If the *old=new* operand is not also specified with -s,
91294 the string form of the *first* operand cannot contain an embedded
91295 <equals-sign>.
- 91296 When the synopsis form with -s is used:
- 91297 If *first* is omitted, the previous command shall be used.
- 91298 For the synopsis forms without -s:
- 91299 If *last* is omitted, *last* shall default to the previous command when -l is
91300 specified; otherwise, it shall default to *first*.
- 91301 If *first* and *last* are both omitted, the previous 16 commands shall be listed or
91302 the previous single command shall be edited (based on the -l option).
- 91303 If *first* and *last* are both present, all of the commands from *first* to *last* shall be
91304 edited (without -l) or listed (with -l). Editing multiple commands shall be
91305 accomplished by presenting to the editor all of the commands at one time,
91306 each command starting on a new line. If *first* represents a newer command
91307 than *last*, the commands shall be listed or edited in reverse sequence,
91308 equivalent to using -r. For example, the following commands on the first
91309 line are equivalent to the corresponding commands on the second:
- 91310 fc -r 10 20 fc 30 40
91311 fc 20 10 fc -r 40 30
- 91312 When a range of commands is used, it shall not be an error to specify *first* or
91313 *last* values that are not in the history list; *fc* shall substitute the value
91314 representing the oldest or newest command in the list, as appropriate. For
91315 example, if there are only ten commands in the history list, numbered 1 to 10:
- 91316 fc -l
91317 fc 1 99
- 91318 shall list and edit, respectively, all ten commands.
- 91319 *old=new* Replace the first occurrence of string *old* in the commands to be re-executed by the
91320 string *new*.

91321 **STDIN**

91322 Not used.

91323 **INPUT FILES**

91324 None.

91325 **ENVIRONMENT VARIABLES**91326 The following environment variables shall affect the execution of *fc*:

91327 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for
 91328 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
 91329 used as the editor.

91330 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is
 91331 not set, the shell may attempt to access or create a file *.sh_history* in the directory
 91332 referred to by the *HOME* environment variable. If the shell cannot obtain both read
 91333 and write access to, or create, the history file, it shall use an unspecified
 91334 mechanism that allows the history to operate properly. (References to history ``file''
 91335 in this section shall be understood to mean this unspecified mechanism in such
 91336 cases.) An implementation may choose to access this variable only when
 91337 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
 91338 to retrieve entries from, or add entries to, the file, as the result of commands issued
 91339 by the user, the file named by the *ENV* variable, or implementation-defined system
 91340 start-up files. In some historical shells, the history file is initialized just after the
 91341 *ENV* file has been processed. Therefore, it is implementation-defined whether
 91342 changes made to *HISTFILE* after the history file has been initialized are effective.
 91343 Implementations may choose to disable the history list mechanism for users with
 91344 appropriate privileges who do not set *HISTFILE*; the specific circumstances under
 91345 which this occurs are implementation-defined. If more than one instance of the
 91346 shell is using the same history file, it is unspecified how updates to the history file
 91347 from those shells interact. As entries are deleted from the history file, they shall be
 91348 deleted oldest first. It is unspecified when history file entries are physically
 91349 removed from the history file.

91350 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
 91351 commands that are accessible. If this variable is unset, an unspecified default
 91352 greater than or equal to 128 shall be used. The maximum number of commands in
 91353 the history list is unspecified, but shall be at least 128. An implementation may
 91354 choose to access this variable only when initializing the history file, as described
 91355 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
 91356 after the history file has been initialized are effective.

91357 *LANG* Provide a default value for the internationalization variables that are unset or null.
 91358 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 91359 variables used to determine the values of locale categories.)

91360 *LC_ALL* If set to a non-empty string value, override the values of all the other
 91361 internationalization variables.

91362 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 91363 characters (for example, single-byte as opposed to multi-byte characters in
 91364 arguments and input files).

91365 *LC_MESSAGES*

91366 Determine the locale that should be used to affect the format and contents of
 91367 diagnostic messages written to standard error.

- 91368 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 91369 **ASYNCHRONOUS EVENTS**
- 91370 Default.
- 91371 **STDOUT**
- 91372 When the **-l** option is used to list commands, the format of each command in the list shall be as follows:
- 91373
- 91374 `"%d\t%s\n", <line number>, <command>`
- 91375 If both the **-l** and **-n** options are specified, the format of each command shall be:
- 91376 `"\t%s\n", <command>`
- 91377 If the *<command>* consists of more than one line, the lines after the first shall be displayed as:
- 91378 `"\t%s\n", <continued-command>`
- 91379 **STDERR**
- 91380 The standard error shall be used only for diagnostic messages.
- 91381 **OUTPUT FILES**
- 91382 None.
- 91383 **EXTENDED DESCRIPTION**
- 91384 None.
- 91385 **EXIT STATUS**
- 91386 The following exit values shall be returned:
- 91387 0 Successful completion of the listing.
- 91388 >0 An error occurred.
- 91389 Otherwise, the exit status shall be that of the commands executed by *fc*.
- 91390 **CONSEQUENCES OF ERRORS**
- 91391 Default.
- 91392 **APPLICATION USAGE**
- 91393 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the *FCEDIT* editor, the command:
- 91394
- 91395 `fc -s | more`
- 91396
- 91397 does not work correctly on many systems.
- 91398 Users on windowing systems may want to have separate history files for each window by setting *HISTFILE* as follows:
- 91399
- 91400 `HISTFILE=$HOME/.sh_hist$$`
- 91401 **EXAMPLES**
- 91402 None.
- 91403 **RATIONALE**
- 91404 This utility is based on the *fc* built-in of the KornShell.
- 91405 An early proposal specified the **-e** option as [**-e** *editor* [*old= new*]], which is not historical practice. Historical practice in *fc* of either [**-e** *editor*] or [**-e** - [*old= new*]] is acceptable, but not both together. To clarify this, a new option **-s** was introduced replacing the [**-e** -]. This resolves
- 91406
- 91407

91408 the conflict and makes *fc* conform to the Utility Syntax Guidelines.

91409 *HISTFILE* Some implementations of the KornShell check for the superuser and do not create
91410 a history file unless *HISTFILE* is set. This is done primarily to avoid creating
91411 unlinked files in the root file system when logging in during single-user mode.
91412 *HISTFILE* must be set for the superuser to have history.

91413 *HISTSIZE* Needed to limit the size of history files. It is the intent of the standard developers
91414 that when two shells share the same history file, commands that are entered in one
91415 shell shall be accessible by the other shell. Because of the difficulties of
91416 synchronization over a network, the exact nature of the interaction is unspecified.

91417 The initialization process for the history file can be dependent on the system start-up files, in
91418 that they may contain commands that effectively preempt the settings the user has for *HISTFILE*
91419 and *HISTSIZE*. For example, function definition commands are recorded in the history file. If
91420 the system administrator includes function definitions in some system start-up file called before
91421 the *ENV* file, the history file is initialized before the user can influence its characteristics. In some
91422 historical shells, the history file is initialized just after the *ENV* file has been processed. Because
91423 of these situations, the text requires the initialization process to be implementation-defined.

91424 Consideration was given to omitting the *fc* utility in favor of the command line editing feature in
91425 *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:

```
91426 EDITOR=vi fc
```

91427 However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously
91428 (such as *fc* 10 20) and to use editors other than those supported by *sh* for command line editing.

91429 In the KornShell, the alias *r* ("re-do") is preset to *fc -e -* (equivalent to the POSIX *fc -s*). This is
91430 probably an easier command name to remember than *fc* ("fix command"), but it does not meet
91431 the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this
91432 description closely matches historical KornShell practice already, such a renaming was seen as
91433 gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*,
91434 *grep*, or *yacc* are standardized by POSIX.

91435 Command numbers have no ordering effects; they are like serial numbers. The *-r* option and
91436 *-number* operand address the sequence of command execution, regardless of serial numbers. So,
91437 for example, if the command number wrapped back to 1 at some arbitrary point, there would be
91438 no ambiguity associated with traversing the wrap point. For example, if the command history
91439 were:

```
91440 32766: echo 1
91441 32767: echo 2
91442 1: echo 3
```

91443 the number *-2* refers to command 32767 because it is the second previous command, regardless
91444 of serial number.

91445 FUTURE DIRECTIONS

91446 None.

91447 SEE ALSO

91448 *sh*

91449 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

91450 **CHANGE HISTORY**

91451 First released in Issue 4.

91452 **Issue 5**

91453 The FUTURE DIRECTIONS section is added.

91454 **Issue 6**

91455 This utility is marked as part of the User Portability Utilities option.

91456 In the ENVIRONMENT VARIABLES section, the text ``user's home directory'' is updated to
91457 ``directory referred to by the *HOME* environment variable''.

91458 **Issue 7**

91459 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91460 **NAME**

91461 fg — run jobs in the foreground

91462 **SYNOPSIS**91463 UP fg [*job_id*]91464 **DESCRIPTION**91465 If job control is enabled (see the description of *set -m*), the *fg* utility shall move a background job
91466 from the current environment (see [Section 2.12](#), on page 2381) into the foreground.91467 Using *fg* to place a job into the foreground shall remove its process ID from the list of those
91468 “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page 2370).91469 **OPTIONS**

91470 None.

91471 **OPERANDS**

91472 The following operand shall be supported:

91473 *job_id* Specify the job to be run as a foreground job. If no *job_id* operand is given, the
91474 *job_id* for the job that was most recently suspended, placed in the background, or
91475 run as a background job shall be used. The format of *job_id* is described in XBD
91476 [Section 3.204](#) (on page 66).91477 **STDIN**

91478 Not used.

91479 **INPUT FILES**

91480 None.

91481 **ENVIRONMENT VARIABLES**91482 The following environment variables shall affect the execution of *fg*:91483 *LANG* Provide a default value for the internationalization variables that are unset or null.
91484 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
91485 variables used to determine the values of locale categories.)91486 *LC_ALL* If set to a non-empty string value, override the values of all the other
91487 internationalization variables.91488 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
91489 characters (for example, single-byte as opposed to multi-byte characters in
91490 arguments).91491 *LC_MESSAGES*91492 Determine the locale that should be used to affect the format and contents of
91493 diagnostic messages written to standard error.91494 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.91495 **ASYNCHRONOUS EVENTS**

91496 Default.

91497 **STDOUT**91498 The *fg* utility shall write the command line of the job to standard output in the following format:

91499 "%s\n", <command>

91500 STDERR

91501 The standard error shall be used only for diagnostic messages.

91502 OUTPUT FILES

91503 None.

91504 EXTENDED DESCRIPTION

91505 None.

91506 EXIT STATUS

91507 The following exit values shall be returned:

91508 0 Successful completion.

91509 >0 An error occurred.

91510 CONSEQUENCES OF ERRORS

91511 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the foreground.
91512

91513 APPLICATION USAGE

91514 The *fg* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell regular built-in.
91515
91516
91517

91518 EXAMPLES

91519 None.

91520 RATIONALE

91521 The extensions to the shell specified in this volume of POSIX.1-2017 have mostly been based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also based on the KornShell. The standard developers examined the characteristics of the C shell versions of these utilities and found that differences exist. Despite widespread use of the C shell, the KornShell versions were selected for this volume of POSIX.1-2017 to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).
91522
91523
91524
91525
91526
91527

91528 FUTURE DIRECTIONS

91529 None.

91530 SEE ALSO

91531 [Section 2.9.3.1](#) (on page 2370), [Section 2.12](#) (on page 2381), *bg*, *kill*, *jobs*, *wait*

91532 XBD [Section 3.204](#) (on page 66), [Chapter 8](#) (on page 173)

91533 CHANGE HISTORY

91534 First released in Issue 4.

91535 Issue 6

91536 This utility is marked as part of the User Portability Utilities option.

91537 The APPLICATION USAGE section is added.

91538 The JC marking is removed from the SYNOPSIS since job control is mandatory in this version.

91539 **NAME**

91540 file ‡determine file type

91541 **SYNOPSIS**91542 file [-dh] [-M *file*] [-m *file*] *file*...91543 file -i [-h] *file*...91544 **DESCRIPTION**91545 The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to
91546 classify it:

- 91547 1. If *file* does not exist, cannot be read, or its file status could not be determined, the output
91548 shall indicate that the file was processed, but that its type could not be determined.
- 91549 2. If the file is not a regular file, its file type shall be identified. The file types directory,
91550 FIFO, socket, block special, and character special shall be identified as such. Other
91551 implementation-defined file types may also be identified. If *file* is a symbolic link, by
91552 default the link shall be resolved and *file* shall test the type of file referenced by the
91553 symbolic link. (See the **-h** and **-i** options below.)
- 91554 3. If the length of *file* is zero, it shall be identified as an empty file.
- 91555 4. The *file* utility shall examine an initial segment of *file* and shall make a guess at
91556 identifying its contents based on position-sensitive tests. (The answer is not guaranteed to
91557 be correct; see the **-d**, **-M**, and **-m** options below.)
- 91558 5. The *file* utility shall examine *file* and make a guess at identifying its contents based on
91559 context-sensitive default system tests. (The answer is not guaranteed to be correct.)
- 91560 6. The file shall be identified as a data file.

91561 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall
91562 indicate that the file was processed, but that its type could not be determined.91563 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file
91564 referenced by the symbolic link.91565 **OPTIONS**91566 The *file* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the order of the **-m**,
91567 **-d**, and **-M** options shall be significant.

91568 The following options shall be supported by the implementation:

- 91569 **-d** Apply any position-sensitive default system tests and context-sensitive default
91570 system tests to the file. This is the default if no **-M** or **-m** option is specified.
- 91571 **-h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is
91572 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall
91573 identify the file as a symbolic link, as if **-h** had been specified.
- 91574 **-i** If a file is a regular file, do not attempt to classify the type of the file further, but
91575 identify the file as specified in the **STDOUT** section.
- 91576 **-M *file*** Specify the name of a file containing position-sensitive tests that shall be applied to
91577 a file in order to classify it (see the **EXTENDED DESCRIPTION**). No position-
91578 sensitive default system tests nor context-sensitive default system tests shall be
91579 applied unless the **-d** option is also specified.

91580 **-m file** Specify the name of a file containing position-sensitive tests that shall be applied to
 91581 a file in order to classify it (see the EXTENDED DESCRIPTION).

91582 If the **-m** option is specified without specifying the **-d** option or the **-M** option, position-
 91583 sensitive default system tests shall be applied after the position-sensitive tests specified by the
 91584 **-m** option. If the **-M** option is specified with the **-d** option, the **-m** option, or both, or the **-m**
 91585 option is specified with the **-d** option, the concatenation of the position-sensitive tests specified
 91586 by these options shall be applied in the order specified by the appearance of these options. If a
 91587 **-M** or **-m file** option-argument is **-**, the results are unspecified.

91588 OPERANDS

91589 The following operand shall be supported:

91590 *file* A pathname of a file to be tested.

91591 STDIN

91592 The standard input shall be used if a *file* operand is **'-'** and the implementation treats the **'-'**
 91593 as meaning standard input. Otherwise, the standard input shall not be used.

91594 INPUT FILES

91595 The *file* can be any file type.

91596 ENVIRONMENT VARIABLES

91597 The following environment variables shall affect the execution of *file*:

91598 *LANG* Provide a default value for the internationalization variables that are unset or null.
 91599 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 91600 variables used to determine the values of locale categories.)

91601 *LC_ALL* If set to a non-empty string value, override the values of all the other
 91602 internationalization variables.

91603 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 91604 characters (for example, single-byte as opposed to multi-byte characters in
 91605 arguments and input files).

91606 *LC_MESSAGES*

91607 Determine the locale that should be used to affect the format and contents of
 91608 diagnostic messages written to standard error and informative messages written to
 91609 standard output.

91610 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

91611 ASYNCHRONOUS EVENTS

91612 Default.

91613 STDOUT

91614 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

91615 "%s: %s\n", <file>, <type>

91616 The values for <type> are unspecified, except that in the POSIX locale, if *file* is identified as one
 91617 of the types listed in the following table, <type> shall contain (but is not limited to) the
 91618 corresponding string, unless the file is identified by a position-sensitive test specified by a **-M** or
 91619 **-m** option. Each <space> shown in the strings shall be exactly one <space>.

91620

Table 4-9 File Utility Output Strings

	If <i>file</i> is:	<<i>type</i>> shall contain the string:	Notes
91621	Nonexistent	cannot open	
91622	Block special	block special	1
91623	Character special	character special	1
91624	Directory	directory	1
91625	FIFO	fifo	1
91626	Socket	socket	1
91627	Symbolic link	symbolic link to	1
91628	Regular file	regular file	1,2
91629	Empty regular file	empty	3
91630	Regular file that cannot be read	cannot open	3
91631	Executable binary	executable	3,4,6
91632	<i>ar</i> archive library (see <i>ar</i>)	archive	3,4,6
91633	Extended <i>cpio</i> format (see <i>pax</i>)	cpio archive	3,4,6
91634	Extended <i>tar</i> format (see ustar in <i>pax</i>)	tar archive	3,4,6
91635	Shell script	commands text	3,5,6
91636	C-language source	c program text	3,5,6
91637	FORTRAN source	fortran program text	3,5,6
91638	Regular file whose type cannot be determined	data	3

91640

Notes:

91641

1. This is a file type test.

91642

2. This test is applied only if the **-i** option is specified.

91643

3. This test is applied only if the **-i** option is not specified.

91644

4. This is a position-sensitive default system test.

91645

5. This is a context-sensitive default system test.

91646

6. Position-sensitive default system tests and context-sensitive default system tests are not applied if the **-M** option is specified unless the **-d** option is also specified.

91647

91648

In the POSIX locale, if *file* is identified as a symbolic link (see the **-h** option), the following alternative output format shall be used:

91649

91650

```
"%s: %s %s\n", <file>, <type>, <contents of link>"
```

91651

If the file named by the *file* operand does not exist, cannot be read, or the type of the file named by the *file* operand cannot be determined, this shall not be considered an error that affects the exit status.

91652

91653

91654

STDERR

91655

The standard error shall be used only for diagnostic messages.

91656

OUTPUT FILES

91657

None.

91658 EXTENDED DESCRIPTION

91659 A file specified as an option-argument to the `-m` or `-M` options shall contain one position-
 91660 sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of
 91661 the line shall be printed and no further tests shall be applied, with the exception that tests on
 91662 immediately following lines beginning with a single `'>'` character shall be applied.

91663 Each line shall be composed of the following four <tab>-separated fields. (Implementations may
 91664 allow any combination of one or more white-space characters other than <newline> to act as
 91665 field separators.)

91666 *offset* An unsigned number (optionally preceded by a single `'>'` character) specifying
 91667 the *offset*, in bytes, of the value in the file that is to be compared against the *value*
 91668 field of the line. If the file is shorter than the specified offset, the test shall fail.

91669 If the *offset* begins with the character `'>'`, the test contained in the line shall not be
 91670 applied to the file unless the test on the last line for which the *offset* did not begin
 91671 with a `'>'` was successful. By default, the *offset* shall be interpreted as an unsigned
 91672 decimal number. With a leading `0x` or `0X`, the *offset* shall be interpreted as a
 91673 hexadecimal number; otherwise, with a leading `0`, the *offset* shall be interpreted as
 91674 an octal number.

91675 *type* The type of the value in the file to be tested. The type shall consist of the type
 91676 specification characters `d`, `s`, and `u`, specifying signed decimal, string, and
 91677 unsigned decimal, respectively.

91678 The *type* string shall be interpreted as the bytes from the file starting at the
 91679 specified *offset* and including the same number of bytes specified by the *value* field.
 91680 If insufficient bytes remain in the file past the *offset* to match the *value* field, the test
 91681 shall fail.

91682 The type specification characters `d` and `u` can be followed by an optional unsigned
 91683 decimal integer that specifies the number of bytes represented by the type. The
 91684 type specification characters `d` and `u` can be followed by an optional `C`, `S`, `I`, or `L`,
 91685 indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

91686 The default number of bytes represented by the type specifiers `d`, `f`, and `u` shall
 91687 correspond to their respective C-language types as follows. If the system claims
 91688 conformance to the C-Language Development Utilities option, those specifiers
 91689 shall correspond to the default sizes used in the *c99* utility. Otherwise, the default
 91690 sizes shall be implementation-defined.

91691 For the type specifier characters `d` and `u`, the default number of bytes shall
 91692 correspond to the size of a basic integer type of the implementation. For these
 91693 specifier characters, the implementation shall support values of the optional
 91694 number of bytes to be converted corresponding to the number of bytes in the C-
 91695 language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an
 91696 application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when
 91697 interpreting numeric values is implementation-defined, but shall correspond to the
 91698 order in which a constant of the corresponding type is stored in memory on the
 91699 system.

91700 All type specifiers, except for `s`, can be followed by a mask specifier of the form
 91701 `&number`. The mask value shall be AND'ed with the value of the input file before
 91702 the comparison with the *value* field of the line is made. By default, the mask shall
 91703 be interpreted as an unsigned decimal number. With a leading `0x` or `0X`, the mask
 91704 shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading

91705		0, the mask shall be interpreted as an unsigned octal number.
91706		The strings byte , short , long , and string shall also be supported as type fields,
91707		being interpreted as dC, dS, dL, and s, respectively.
91708	<i>value</i>	The <i>value</i> to be compared with the value from the file.
91709		If the specifier from the type field is s or string , then interpret the value as a string.
91710		Otherwise, interpret it as a number. If the value is a string, then the test shall
91711		succeed only when a string value exactly matches the bytes from the file.
91712		If the <i>value</i> is a string, it can contain the following sequences:
91713	<i>\character</i>	The <backslash>-escape sequences as specified in XBD Table 5-1
91714		(on page 121) ('\a', '\b', '\f', '\n', '\r', '\t',
91715		'\v'). In addition, the escape sequence '\ ' (the <backslash>
91716		character followed by a <space> character) shall be recognized to
91717		represent a <space> character. The results of using any other
91718		character, other than an octal digit, following the <backslash>
91719		are unspecified.
91720	<i>\octal</i>	Octal sequences that can be used to represent characters with
91721		specific coded values. An octal sequence shall consist of a
91722		<backslash> followed by the longest sequence of one, two, or
91723		three octal-digit characters (01234567).
91724		By default, any value that is not a string shall be interpreted as a signed decimal
91725		number. Any such value, with a leading 0x or 0X, shall be interpreted as an
91726		unsigned hexadecimal number; otherwise, with a leading zero, the value shall be
91727		interpreted as an unsigned octal number.
91728		If the value is not a string, it can be preceded by a character indicating the
91729		comparison to be performed. Permissible characters and the comparisons they
91730		specify are as follows:
91731	=	The test shall succeed if the value from the file equals the <i>value</i> field.
91732	<	The test shall succeed if the value from the file is less than the <i>value</i> field.
91733	>	The test shall succeed if the value from the file is greater than the <i>value</i> field.
91734	&	The test shall succeed if all of the set bits in the <i>value</i> field are set in the value
91735		from the file.
91736	^	The test shall succeed if at least one of the set bits in the <i>value</i> field is not set in
91737		the value from the file.
91738	x	The test shall succeed if the file is large enough to contain a value of the type
91739		specified starting at the offset specified.
91740	<i>message</i>	The <i>message</i> to be printed if the test succeeds. The <i>message</i> shall be interpreted
91741		using the notation for the <i>printf</i> formatting specification; see <i>printf</i> . If the <i>value</i>
91742		field was a string, then the value from the file shall be the argument for the <i>printf</i>
91743		formatting specification; otherwise, the value from the file shall be the argument.

91744 **EXIT STATUS**

91745 The following exit values shall be returned:

91746 0 Successful completion.

91747 >0 An error occurred.

91748 **CONSEQUENCES OF ERRORS**

91749 Default.

91750 **APPLICATION USAGE**91751 The *file* utility can only be required to guess at many of the file types because only exhaustive
91752 testing can determine some types with certainty. For example, binary data on some
91753 implementations might match the initial segment of an executable or a *tar* archive.91754 Note that the table indicates that the output contains the stated string. Systems may add text
91755 before or after the string. For executables, as an example, the machine architecture and various
91756 facts about how the file was link-edited may be included. Note also that on systems that
91757 recognize shell script files starting with "#!" as executable files, these may be identified as
91758 executable binary files rather than as shell scripts.91759 **EXAMPLES**

91760 Determine whether an argument is a binary executable file:

91761

```
file -- "$1" | grep -q '.*executable' &&  
91762     printf "%s is executable.\n" "$1"
```

91763 **RATIONALE**91764 The **-f** option was omitted because the same effect can (and should) be obtained using the *xargs*
91765 utility.91766 Historical versions of the *file* utility attempt to identify the following types of files: symbolic link,
91767 directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive
91768 library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler
91769 source, *nroff*/*troff*/*eqn*/*tbl* source *troff* output, shell script, C shell script, English text, ASCII text,
91770 various executables, APL workspace, compiled terminfo entries, and CURSES screen images.
91771 Only those types that are reasonably well specified in POSIX or are directly related to POSIX
91772 utilities are listed in the table.91773 Historical systems have used a "magic file" named */etc/magic* to help identify file types. Because
91774 it is generally useful for users and scripts to be able to identify special file types, the **-m** flag and
91775 a portable format for user-created magic files has been specified. No requirement is made that an
91776 implementation of *file* use this method of identifying files, only that users be permitted to add
91777 their own classifying tests.91778 In addition, three options have been added to historical practice. The **-d** flag has been added to
91779 permit users to cause their tests to follow any default system tests. The **-i** flag has been added to
91780 permit users to test portably for regular files in shell scripts. The **-M** flag has been added to
91781 permit users to ignore any default system tests.91782 The POSIX.1-2017 description of default system tests and the interaction between the **-d**, **-M**,
91783 and **-m** options did not clearly indicate that there were two types of "default system tests". The
91784 "position-sensitive tests" determine file types by looking for certain string or binary values at
91785 specific offsets in the file being examined. These position-sensitive tests were implemented in
91786 historical systems using the magic file described above. Some of these tests are now built into
91787 the *file* utility itself on some implementations so the output can provide more detail than can be
91788 provided by magic files. For example, a magic file can easily identify a **core** file on most
91789 implementations, but cannot name the program file that dropped the core. A magic file could

91790 produce output such as:

91791 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1

91792 but by building the test into the *file* utility, you could get output such as:

91793 /home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'

91794 These extended built-in tests are still to be treated as position-sensitive default system tests even
91795 if they are not listed in */etc/magic* or any other magic file.

91796 The context-sensitive default system tests were always built into the *file* utility. These tests
91797 looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and
91798 other computer language source files, and even plain text files. With the addition of the *-m* and
91799 *-M* options the distinction between position-sensitive and context-sensitive default system tests
91800 became important because the order of testing is important. The context-sensitive system default
91801 tests should never be applied before any position-sensitive tests even if the *-d* option is specified
91802 before a *-m* option or *-M* option due to the high probability that the context-sensitive system
91803 default tests will incorrectly identify arbitrary text files as text files before position-sensitive tests
91804 specified by the *-m* or *-M* option would be applied to give a more accurate identification.

91805 Leaving the meaning of *-M* – and *-m* – unspecified allows an existing prototype of these
91806 options to continue to work in a backwards-compatible manner. (In that implementation, *-M* –
91807 was roughly equivalent to *-d* in POSIX.1-2017.)

91808 The historical *-c* option was omitted as not particularly useful to users or portable shell scripts.
91809 In addition, a reasonable implementation of the *file* utility would report any errors found each
91810 time the magic file is read.

91811 The historical format of the magic file was the same as that specified by the Rationale in the
91812 ISO POSIX-2:1993 standard for the *offset*, *value*, and *message* fields; however, it used less precise
91813 type fields than the format specified by the current normative text. The new type field values are
91814 a superset of the historical ones.

91815 The following is an example magic file:

```
91816 0 short      070707          cpio archive
91817 0 short      0143561        Byte-swapped cpio archive
91818 0 string     070707          ASCII cpio archive
91819 0 long       0177555        Very old archive
91820 0 short      0177545        Old archive
91821 0 short      017437         Old packed data
91822 0 string     \037\036       Packed data
91823 0 string     \377\037       Compacted data
91824 0 string     \037\235       Compressed data
91825 >2 byte&0x80 >0          Block compressed
91826 >2 byte&0x1f x           %d bits
91827 0 string     \032\001       Compiled Terminfo Entry
91828 0 short      0433           Curses screen image
91829 0 short      0434           Curses screen image
91830 0 string     <ar>           System V Release 1 archive
91831 0 string     !<arch>\n___.SYMDEF Archive random library
91832 0 string     !<arch>        Archive
91833 0 string     ARF_BEGARF     PHIGS clear text archive
91834 0 long       0x137A2950     Scalable OpenFont binary
91835 0 long       0x137A2951     Encrypted scalable OpenFont binary
```

- 91836 The use of a basic integer data type is intended to allow the implementation to choose a word
91837 size commonly used by applications on that architecture.
- 91838 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
91839 but this has been modified in this version.
- 91840 **FUTURE DIRECTIONS**
- 91841 None.
- 91842 **SEE ALSO**
- 91843 *ar, ls, pax, printf*
- 91844 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 91845 **CHANGE HISTORY**
- 91846 First released in Issue 4.
- 91847 **Issue 6**
- 91848 This utility is marked as part of the User Portability Utilities option.
- 91849 Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft
91850 standard.
- 91851 IEEE PASC Interpretations 1003.2 #192 and #178 are applied.
- 91852 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to
91853 address ambiguities raised in defect reports.
- 91854 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the
91855 OPTIONS section that the **-m**, **-d**, and **-M** options do not comply with Guideline 11 of the
91856 Utility Syntax Guidelines.
- 91857 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/10 is applied, clarifying the specification
91858 characters.
- 91859 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/11 is applied, allowing application
91860 developers to create portable magic files that can match characters in strings, and allowing
91861 common extensions found in existing implementations.
- 91862 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/12 is applied, removing text describing
91863 behavior on systems with bytes consisting of more than eight bits.
- 91864 **Issue 7**
- 91865 Austin Group Interpretation 1003.1-2001 #092 is applied.
- 91866 SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in [Table 4-9](#).
- 91867 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 91868 The *file* utility is moved from the User Portability Utilities option to the Base. User Portability
91869 Utilities is now an option for interactive utilities.
- 91870 The EXAMPLES section is revised to correct an error with the pathname "\$1".

91871 **NAME**

91872 find ‡find files

91873 **SYNOPSIS**91874 find [-H|-L] *path...* [*operand_expression...*]91875 **DESCRIPTION**

91876 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,
 91877 evaluating a Boolean expression composed of the primaries described in the OPERANDS section
 91878 for each file encountered. Each *path* operand shall be evaluated unaltered as it was provided,
 91879 including all trailing <slash> characters; all pathnames for other files encountered in the
 91880 hierarchy shall consist of the concatenation of the current *path* operand, a <slash> if the current
 91881 *path* operand did not end in one, and the filename relative to the *path* operand. The relative
 91882 portion shall contain no dot or dot-dot components, no trailing <slash> characters, and only
 91883 single <slash> characters between pathname components.

91884 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail
 91885 due to path length limitations (unless a *path* operand specified by the application exceeds
 91886 {PATH_MAX} requirements).

91887 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an
 91888 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a
 91889 diagnostic message to standard error and shall either recover its position in the hierarchy or
 91890 terminate.

91891 If a file is removed from or added to the directory hierarchy being searched it is unspecified
 91892 whether or not *find* includes that file in its search.

91893 **OPTIONS**91894 The *find* utility shall conform to XBD [Section 12.2](#) (on page 216).

91895 The following options shall be supported by the implementation:

91896 **-H** Cause the file information and file type evaluated for each symbolic link
 91897 encountered as a *path* operand on the command line to be those of the file
 91898 referenced by the link, and not the link itself. If the referenced file does not exist,
 91899 the file information and type shall be for the link itself. File information and type
 91900 for symbolic links encountered during the traversal of a file hierarchy shall be that
 91901 of the link itself.

91902 **-L** Cause the file information and file type evaluated for each symbolic link
 91903 encountered as a *path* operand on the command line or encountered during the
 91904 traversal of a file hierarchy to be those of the file referenced by the link, and not the
 91905 link itself. If the referenced file does not exist, the file information and type shall be
 91906 for the link itself.

91907 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
 91908 an error. The last option specified shall determine the behavior of the utility. If neither the **-H**
 91909 nor the **-L** option is specified, then the file information and type for symbolic links encountered
 91910 as a *path* operand on the command line or encountered during the traversal of a file hierarchy
 91911 shall be that of the link itself.

91912 **OPERANDS**

91913 The following operands shall be supported:

91914 The first operand and subsequent operands up to but not including the first operand that starts
 91915 with a '-', or is a '!' or a '(', shall be interpreted as *path* operands. If the first operand starts
 91916 with a '-', or is a '!' or a '(', the behavior is unspecified. Each *path* operand is a pathname of

91917 a starting point in the file hierarchy.

91918 The first operand that starts with a '-', or is a '!' or a '(' , and all subsequent arguments shall
 91919 be interpreted as an *expression* made up of the following primaries and operators. In the
 91920 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal
 91921 integer optionally preceded by a <plus-sign> ('+') or <hyphen-minus> ('-'), as follows:

91922 **+n** More than *n*.

91923 **n** Exactly *n*.

91924 **-n** Less than *n*.

91925 The following primaries shall be supported:

91926 **-name pattern**

91927 The primary shall evaluate as true if the basename of the current pathname
 91928 matches *pattern* using the pattern matching notation described in [Section 2.13](#) (on
 91929 page 2382). The additional rules in [Section 2.13.3](#) (on page 2383) do not apply as
 91930 this is a matching operation, not an expansion.

91931 **-path pattern**

91932 The primary shall evaluate as true if the current pathname matches *pattern* using
 91933 the pattern matching notation described in [Section 2.13](#) (on page 2382). The
 91934 additional rules in [Section 2.13.3](#) (on page 2383) do not apply as this is a matching
 91935 operation, not an expansion.

91936 **-nouser** The primary shall evaluate as true if the file belongs to a user ID for which the
 91937 *getpwuid()* function defined in the System Interfaces volume of POSIX.1-2017 (or
 91938 equivalent) returns NULL.

91939 **-nogroup** The primary shall evaluate as true if the file belongs to a group ID for which the
 91940 *getgrgid()* function defined in the System Interfaces volume of POSIX.1-2017 (or
 91941 equivalent) returns NULL.

91942 **-xdev** The primary shall always evaluate as true; it shall cause *find* not to continue
 91943 descending past directories that have a different device ID (*st_dev*, see the *stat()*
 91944 function defined in the System Interfaces volume of POSIX.1-2017). If any **-xdev**
 91945 primary is specified, it shall apply to the entire expression even if the **-xdev**
 91946 primary would not normally be evaluated.

91947 **-prune** The primary shall always evaluate as true; it shall cause *find* not to descend the
 91948 current pathname if it is a directory. If the **-depth** primary is specified, the **-prune**
 91949 primary shall have no effect.

91950 **-perm [-]mode**

91951 The *mode* argument is used to represent file mode bits. It shall be identical in
 91952 format to the *symbolic_mode* operand described in *chmod*, and shall be interpreted
 91953 as follows. To start, a template shall be assumed with all file mode bits cleared. An
 91954 *op* symbol of '+' shall set the appropriate mode bits in the template; '-' shall
 91955 clear the appropriate bits; '=' shall set the appropriate mode bits, without regard
 91956 to the contents of the file mode creation mask of the process. The *op* symbol of '-'
 91957 cannot be the first character of *mode*; this avoids ambiguity with the optional
 91958 leading <hyphen-minus>. Since the initial mode is all bits off, there are not any
 91959 symbolic modes that need to use '-' as the first character.

91960 If the <hyphen-minus> is omitted, the primary shall evaluate as true when the file
 91961 permission bits exactly match the value of the resulting template.

91962 Otherwise, if *mode* is prefixed by a <hyphen-minus>, the primary shall evaluate as
 91963 true if at least all the bits in the resulting template are set in the file permission bits.

91964 **-perm** [-]*onum*
 91965 If the <hyphen-minus> is omitted, the primary shall evaluate as true when the file
 91966 mode bits exactly match the value of the octal number *onum* (see the description of
 91967 the octal *mode* in *chmod*). Otherwise, if *onum* is prefixed by a <hyphen-minus>, the
 91968 primary shall evaluate as true if at least all of the bits specified in *onum* are set. In
 91969 both cases, the behavior is unspecified when *onum* exceeds 07777.

91970 **-type** *c* The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c',
 91971 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory,
 91972 symbolic link, FIFO, regular file, or socket, respectively.

91973 **-links** *n* The primary shall evaluate as true if the file has *n* links.

91974 **-user** *uname* The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is
 91975 a decimal integer and the *getpwnam()* (or equivalent) function does not return a
 91976 valid user name, *uname* shall be interpreted as a user ID.

91977 **-group** *gname*
 91978 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*
 91979 is a decimal integer and the *getgrnam()* (or equivalent) function does not return a
 91980 valid group name, *gname* shall be interpreted as a group ID.

91981 **-size** *n*[*c*] The primary shall evaluate as true if the file size in bytes, divided by 512 and
 91982 rounded up to the next integer, is *n*. If *n* is followed by the character '*c*', the size
 91983 shall be in bytes.

91984 **-atime** *n* The primary shall evaluate as true if the file access time subtracted from the
 91985 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

91986 **-ctime** *n* The primary shall evaluate as true if the time of last change of file status
 91987 information subtracted from the initialization time, divided by 86 400 (with any
 91988 remainder discarded), is *n*.

91989 **-mtime** *n* The primary shall evaluate as true if the file modification time subtracted from the
 91990 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

91991 **-exec** *utility_name* [*argument* ...];
 91992 **-exec** *utility_name* [*argument* ...] {} +
 91993 The end of the primary expression shall be punctuated by a <semicolon> or by a
 91994 <plus-sign>. Only a <plus-sign> that immediately follows an argument
 91995 containing only the two characters "{}" shall punctuate the end of the primary
 91996 expression. Other uses of the <plus-sign> shall not be treated as special.

91997 If the primary expression is punctuated by a <semicolon>, the utility *utility_name*
 91998 shall be invoked once for each pathname and the primary shall evaluate as true if
 91999 the utility returns a zero value as exit status. A *utility_name* or *argument* containing
 92000 only the two characters "{}" shall be replaced by the current pathname. If a
 92001 *utility_name* or *argument* string contains the two characters "{}", but not just the
 92002 two characters "{}", it is implementation-defined whether *find* replaces those two
 92003 characters or uses the string without change.

92004 If the primary expression is punctuated by a <plus-sign>, the primary shall always
 92005 evaluate as true, and the pathnames for which the primary is evaluated shall be
 92006 aggregated into sets. The utility *utility_name* shall be invoked once for each set of
 92007 aggregated pathnames. Each invocation shall begin after the last pathname in the

92008 set is aggregated, and shall be completed before the *find* utility exits and before the
 92009 first pathname in the next set (if any) is aggregated for this primary, but it is
 92010 otherwise unspecified whether the invocation occurs before, during, or after the
 92011 evaluations of other primaries. If any invocation returns a non-zero value as exit
 92012 status, the *find* utility shall return a non-zero exit status. An argument containing
 92013 only the two characters "{}" shall be replaced by the set of aggregated
 92014 pathnames, with each pathname passed as a separate argument to the invoked
 92015 utility in the same order that it was aggregated. The size of any set of two or more
 92016 pathnames shall be limited such that execution of the utility does not cause the
 92017 system's {ARG_MAX} limit to be exceeded. If more than one argument containing
 92018 the two characters "{}" is present, the behavior is unspecified.

92019 The current directory for the invocation of *utility_name* shall be the same as the
 92020 current directory when the *find* utility was started. If the *utility_name* names any of
 92021 the special built-in utilities (see Section 2.14, on page 2384), the results are
 92022 undefined.

92023 **-ok** *utility_name* [*argument* ...];

92024 The **-ok** primary shall be equivalent to **-exec**, except that the use of a <plus-sign>
 92025 to punctuate the end of the primary expression need not be supported, and *find*
 92026 shall request affirmation of the invocation of *utility_name* using the current file as
 92027 an argument by writing to standard error as described in the STDERR section. If
 92028 the response on standard input is affirmative, the utility shall be invoked.
 92029 Otherwise, the command shall not be invoked and the value of the **-ok** operand
 92030 shall be false.

92031 **-print** The primary shall always evaluate as true; it shall cause the current pathname to
 92032 be written to standard output.

92033 **-newer** *file* The primary shall evaluate as true if the modification time of the current file is
 92034 more recent than the modification time of the file named by the pathname *file*.

92035 **-depth** The primary shall always evaluate as true; it shall cause descent of the directory
 92036 hierarchy to be done so that all entries in a directory are acted on before the
 92037 directory itself. If a **-depth** primary is not specified, all entries in a directory shall
 92038 be acted on after the directory itself. If any **-depth** primary is specified, it shall
 92039 apply to the entire expression even if the **-depth** primary would not normally be
 92040 evaluated.

92041 The primaries can be combined using the following operators (in order of decreasing
 92042 precedence):

92043 (*expression*) True if *expression* is true.

92044 **!***expression* Negation of a primary; the unary NOT operator.

92045 *expression* [**-a**]*expression*

92046 Conjunction of primaries; the AND operator is implied by the juxtaposition of two
 92047 primaries or made explicit by the optional **-a** operator. The second expression shall
 92048 not be evaluated if the first expression is false.

92049 *expression* **-o** *expression*

92050 Alternation of primaries; the OR operator. The second expression shall not be
 92051 evaluated if the first expression is true.

92052 If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given
 92053 expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression

- 92054 shall be effectively replaced by:
- 92055 (*given_expression*) -print
- 92056 The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only
92057 once.
- 92058 When the file type evaluated for the current file is a symbolic link, the results of evaluating the
92059 **-perm** primary are implementation-defined.
- 92060 **STDIN**
- 92061 If the **-ok** primary is used, the response shall be read from the standard input. An entire line
92062 shall be read as the response. Otherwise, the standard input shall not be used.
- 92063 **INPUT FILES**
- 92064 None.
- 92065 **ENVIRONMENT VARIABLES**
- 92066 The following environment variables shall affect the execution of *find*:
- 92067 **LANG** Provide a default value for the internationalization variables that are unset or null.
92068 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
92069 variables used to determine the values of locale categories.)
- 92070 **LC_ALL** If set to a non-empty string value, override the values of all the other
92071 internationalization variables.
- 92072 **LC_COLLATE**
- 92073 Determine the locale for the behavior of ranges, equivalence classes, and multi-
92074 character collating elements used in the pattern matching notation for the **-n**
92075 option and in the extended regular expression defined for the **yesexpr** locale
92076 keyword in the *LC_MESSAGES* category.
- 92077 **LC_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of
92078 text data as characters (for example, single-byte as opposed to multi-byte
92079 characters in arguments), the behavior of character classes within the pattern
92080 matching notation used for the **-n** option, and the behavior of character classes
92081 within regular expressions used in the extended regular expression defined for the
92082 **yesexpr** locale keyword in the *LC_MESSAGES* category.
- 92083 **LC_MESSAGES**
- 92084 Determine the locale used to process affirmative responses, and the locale used to
92085 affect the format and contents of diagnostic messages and prompts written to
92086 standard error.
- 92087 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 92088 **PATH** Determine the location of the *utility_name* for the **-exec** and **-ok** primaries, as
92089 described in XBD [Chapter 8](#) (on page 173).
- 92090 **ASYNCHRONOUS EVENTS**
- 92091 Default.
- 92092 **STDOUT**
- 92093 The **-print** primary shall cause the current pathnames to be written to standard output. The
92094 format shall be:
- 92095 "%s\n", <path>

92096 **STDERR**

92097 The **-ok** primary shall write a prompt to standard error containing at least the *utility_name* to be
 92098 invoked and the current pathname. In the POSIX locale, the last non-`<blank>` in the prompt shall
 92099 be `'?'`. The exact format used is unspecified.

92100 Otherwise, the standard error shall be used only for diagnostic messages.

92101 **OUTPUT FILES**

92102 None.

92103 **EXTENDED DESCRIPTION**

92104 None.

92105 **EXIT STATUS**

92106 The following exit values shall be returned:

92107 0 All *path* operands were traversed successfully.

92108 >0 An error occurred.

92109 **CONSEQUENCES OF ERRORS**

92110 Default.

92111 **APPLICATION USAGE**

92112 When used in operands, pattern matching notation, `<semicolon>`, `<left-parenthesis>`, and
 92113 `<right-parenthesis>` characters are special to the shell and must be quoted (see [Section 2.2](#), on
 92114 page 2346).

92115 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary
 92116 using the octal number argument form. Since this bit is not defined by this volume of
 92117 POSIX.1-2017, applications must not assume that it actually refers to the traditional sticky bit.

92118 **EXAMPLES**

92119 1. The following commands are equivalent:

```
92120 find .
92121 find . -print
```

92122 They both write out the entire directory hierarchy from the current directory.

92123 2. The following command:

```
92124 find / \( -name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;
```

92125 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or
 92126 more 24-hour periods.

92127 3. The following command:

```
92128 find . -perm -o+w,+s
```

92129 prints (**-print** is assumed) the names of all files in or below the current directory, with all
 92130 of the file permission bits `S_ISUID`, `S_ISGID`, and `S_IWOTH` set.

92131 4. The following command:

```
92132 find . -name SCCS -prune -o -print
```

92133 recursively prints pathnames of all files in the current directory and below, but skips
 92134 directories named **SCCS** and files in them.

- 92135 5. The following command:
- 92136 `find . -print -name SCCS -prune`
- 92137 behaves as in the previous example, but prints the names of the SCCS directories.
- 92138 6. The following command is roughly equivalent to the `-nt` extension to `test`:
- 92139 `if [-n "$(find file1 -prune -newer file2)"]; then`
- 92140 `printf %s\\n "file1 is newer than file2"`
- 92141 `fi`
- 92142 7. The descriptions of `-atime`, `-ctime`, and `-mtime` use the terminology *n* “86 400 second
- 92143 periods (days)”. For example, a file accessed at 23:59 is selected by:
- 92144 `find . -atime -1 -print`
- 92145 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight
- 92146 boundary between days has no effect on the 24-hour calculation.
- 92147 8. The following command:
- 92148 `find . ! -name . -prune -name '*.old' -exec \`
- 92149 `sh -c 'mv "$@" ../old/' sh {} +`
- 92150 performs the same task as:
- 92151 `mv /*.old ../old /*.old ../old/`
- 92152 while avoiding an “Argument list too long” error if there are a large number of files
- 92153 ending with `.old` and without running `mv` if there are no such files (and avoiding “No
- 92154 such file or directory” errors if `.old` does not exist or no files match `/*.old` or `/*.old`).
- 92155 The alternative:
- 92156 `find . ! -name . -prune -name '*.old' -exec mv {} ../old/ \;`
- 92157 is less efficient if there are many files to move because it executes one `mv` command per
- 92158 file.
- 92159 9. On systems configured to mount removable media on directories under `/media`, the
- 92160 following command searches the file hierarchy for files larger than 100 000 KB without
- 92161 searching any mounted removable media:
- 92162 `find / -path /media -prune -o -size +200000 -print`
- 92163 10. Except for the root directory, and `"/"` on implementations where `"/"` does not refer to
- 92164 the root directory, no pattern given to `-name` will match a `<slash>`, because trailing
- 92165 `<slash>` characters are ignored when computing the basename of the file under
- 92166 evaluation. Given two empty directories named `foo` and `bar`, the following command:
- 92167 `find foo/// bar/// -name foo -o -name 'bar?*'`
- 92168 prints only the line `foo///`.

92169 RATIONALE

92170 The `-a` operator was retained as an optional operator for compatibility with historical shell

92171 scripts, even though it is redundant with expression concatenation.

92172 The descriptions of the `'-'` modifier on the `mode` and `onum` arguments to the `-perm` primary

92173 agree with historical practice on BSD and System V implementations. System V and BSD

92174 documentation both describe it in terms of checking additional bits; in fact, it uses the same bits,

92175 but checks for having at least all of the matching bits set instead of having exactly the matching

92176 bits set.

92177 The exact format of the interactive prompts is unspecified. Only the general nature of the
92178 contents of prompts are specified because:

92179 Implementations may desire more descriptive prompts than those used on historical
92180 implementations.

92181 Since the historical prompt strings do not terminate with <newline> characters, there is no
92182 portable way for another program to interact with the prompts of this utility via pipes.

92183 Therefore, an application using this prompting option relies on the system to provide the most
92184 suitable dialog directly with the user, based on the general guidelines specified.

92185 The **-name** *file* operand was changed to use the shell pattern matching notation so that *find* is
92186 consistent with other utilities using pattern matching.

92187 The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in
92188 the file system. The intent is that the *st_size* field defined in the System Interfaces volume of
92189 POSIX.1-2017 should be used, not the *st_blocks* found in historical implementations. There are at
92190 least two reasons for this:

- 92191 1. In both System V and BSD, *find* only uses *st_size* in size calculations for the operands
92192 specified by this volume of POSIX.1-2017. (BSD uses *st_blocks* only when processing the
92193 **-ls** primary.)
- 92194 2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls*
92195 utility for the output from the **-l** option. (In both System V and BSD, *ls* uses *st_size* for the
92196 **-l** option size field and uses *st_blocks* for the *ls -s* calculations. This volume of
92197 POSIX.1-2017 does not specify *ls -s*.)

92198 The descriptions of **-atime**, **-ctime**, and **-mtime** were changed from the SVID description of *n*
92199 ``days'' to *n* being the result of the integer division of the time difference in seconds by 86 400.
92200 The description is also different in terms of the exact timeframe for the *n* case (*versus* the *+n* or
92201 *-n*), but it matches all known historical implementations. It refers to one 86 400 second period in
92202 the past, not any time from the beginning of that period to the current time. For example, **-atime**
92203 2 is true if the file was accessed any time in the period from 72 hours to 48 hours ago.

92204 Historical implementations do not modify "{ }" when it appears as a substring of an **-exec** or
92205 **-ok** *utility_name* or argument string. There have been numerous user requests for this extension,
92206 so this volume of POSIX.1-2017 allows the desired behavior. At least one recent implementation
92207 does support this feature, but encountered several problems in managing memory allocation
92208 and dealing with multiple occurrences of "{ }" in a string while it was being developed, so it is
92209 not yet required behavior.

92210 Assuming the presence of **-print** was added to correct a historical pitfall that plagues novice
92211 users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest
92212 form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers
92213 agreed that adding **-print** as a default expression was the correct decision and have added the
92214 fast *find* functionality within a new utility called *locate*.

92215 Historically, the **-L** option was implemented using the primary **-follow**. The **-H** and **-L** options
92216 were added for two reasons. First, they offer a finer granularity of control and consistency with
92217 other programs that walk file hierarchies. Second, the **-follow** primary always evaluated to true.
92218 As they were historically really global variables that took effect before the traversal began, some
92219 valid expressions had unexpected results. An example is the expression **-print -o -follow**.
92220 Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow**
92221 would never be evaluated. This was never the case. Historical practice for the **-follow** primary,

92222 however, is not consistent. Some implementations always follow symbolic links on the
 92223 command line whether **-follow** is specified or not. Others follow symbolic links on the
 92224 command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L**
 92225 options, but scripts using the current **-follow** primary would be broken if the **-follow** option is
 92226 specified to work either way.

92227 Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links
 92228 that still exist after symbolic links have been resolved, the command:

```
92229 find -L . -type l
```

92230 prints a list of symbolic links reachable from the current directory that do not resolve to
 92231 accessible files.

92232 A feature of SVR4's *find* utility was the **-exec** primary's **+** terminator. This allowed filenames
 92233 containing special characters (especially <newline> characters) to be grouped together without
 92234 the problems that occur if such filenames are piped to *xargs*. Other implementations have added
 92235 other ways to get around this problem, notably a **-print0** primary that wrote filenames with a
 92236 null byte terminator. This was considered here, but not adopted. Using a null terminator meant
 92237 that any utility that was going to process *find*'s **-print0** output had to add a new option to parse
 92238 the null terminators it would now be reading.

92239 The **"-exec ... {} +"** syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210.
 92240 It should be noted that this is an incompatible change to IEEE Std 1003.2-1992. For example, the
 92241 following command printed all files with a '-' after their name if they are regular files, and a
 92242 '+' otherwise:

```
92243 find / -type f -exec echo {} - ';' -o -exec echo {} + ';' -
```

92244 The change invalidates usage like this. Even though the previous standard stated that this usage
 92245 would work, in practice many did not support it and the standard developers felt it better to
 92246 now state that this was not allowable.

92247 FUTURE DIRECTIONS

92248 None.

92249 SEE ALSO

92250 [Section 2.2](#) (on page 2346), [Section 2.13](#) (on page 2382), [Section 2.14](#) (on page 2384), *chmod*, *mv*,
 92251 *pax*, *sh*, *test*

92252 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

92253 XSH *fstatat()*, *getgrgid()*, *getpwuid()*

92254 CHANGE HISTORY

92255 First released in Issue 2.

92256 Issue 5

92257 The FUTURE DIRECTIONS section is added.

92258 Issue 6

92259 The following new requirements on POSIX implementations derive from alignment with the
 92260 Single UNIX Specification:

92261 The **-perm [-]onum** primary is supported.

92262 The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of
 92263 symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

92264 IEEE PASC Interpretation 1003.2 #210 is applied, extending the **-exec** operand.

- 92265 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/13 is applied, updating the RATIONALE
92266 section to be consistent with the normative text.
- 92267 **Issue 7**
- 92268 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
92269 `LC_MESSAGES` environment variable.
- 92270 Austin Group Interpretation 1003.1-2001 #127 is applied, rephrasing the description of the `-exec`
92271 primary to be “immediately follows”.
- 92272 Austin Group Interpretation 1003.1-2001 #185 is applied, clarifying the requirements for the `-H`
92273 and `-L` options.
- 92274 Austin Group Interpretation 1003.1-2001 #186 is applied, clarifying the requirements for the
92275 evaluation of *path* operands.
- 92276 Austin Group Interpretation 1003.1-2001 #195 is applied, clarifying the interpretation of the first
92277 operand.
- 92278 SD5-XCU-ERN-48 is applied, clarifying the `-L` option in the case that the referenced file does not
92279 exist.
- 92280 SD5-XCU-ERN-89 is applied, updating the OPERANDS section.
- 92281 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 92282 SD5-XCU-ERN-117 is applied, clarifying the `-perm` operand.
- 92283 SD5-XCU-ERN-122 is applied, adding a new EXAMPLE.
- 92284 The description of the `-name` primary is revised and the `-path` primary is added (with a new
92285 example).
- 92286 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0086 [365], XCU/TC1-2008/0087
92287 [310], XCU/TC1-2008/0088 [309,310,430], XCU/TC1-2008/0089 [235], and XCU/TC1-2008/0090
92288 [445] are applied.
- 92289 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0099 [584], XCU/TC2-2008/0100
92290 [584], and XCU/TC2-2008/0101 [584] are applied.

92291 **NAME**

92292 fold ‡filter for folding lines

92293 **SYNOPSIS**92294 fold [-bs] [-w *width*] [*file...*]92295 **DESCRIPTION**

92296 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a
 92297 maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be
 92298 broken by the insertion of a <newline> such that each output line (referred to later in this section
 92299 as a *segment*) is the maximum width possible that does not exceed the specified number of
 92300 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior
 92301 is undefined if *width* is less than the number of columns any single character in the input would
 92302 occupy.

92303 If the <carriage-return>, <backspace>, or <tab> characters are encountered in the input, and the
 92304 **-b** option is not specified, they shall be treated specially:

92305 <backspace> The current count of line width shall be decremented by one, although the count
 92306 never shall become negative. The *fold* utility shall not insert a <newline>
 92307 immediately before or after any <backspace>, unless the following character has a
 92308 width greater than 1 and would cause the line width to exceed *width*.

92309 <carriage-return>
 92310 The current count of line width shall be set to zero. The *fold* utility shall not insert a
 92311 <newline> immediately before or after any <carriage-return>.

92312 <tab> Each <tab> encountered shall advance the column position pointer to the next tab
 92313 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

92314 **OPTIONS**92315 The *fold* utility shall conform to XBD [Section 12.2](#) (on page 216).

92316 The following options shall be supported:

92317 **-b** Count *width* in bytes rather than column positions.

92318 **-s** If a segment of a line contains a <blank> within the first *width* column positions (or
 92319 bytes), break the line after the last such <blank> meeting the width constraints. If
 92320 there is no <blank> meeting the requirements, the **-s** option shall have no effect for
 92321 that output segment of the input line.

92322 **-w *width*** Specify the maximum line length, in column positions (or bytes if **-b** is specified).
 92323 The results are unspecified if *width* is not a positive decimal number. The default
 92324 value shall be 80.

92325 **OPERANDS**

92326 The following operand shall be supported:

92327 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard
 92328 input shall be used.

92329 **STDIN**

92330 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
 92331 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
 92332 the standard input shall not be used. See the INPUT FILES section.

92333 **INPUT FILES**

92334 If the **-b** option is specified, the input files shall be text files except that the lines are not limited
 92335 to {LINE_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

92336 **ENVIRONMENT VARIABLES**

92337 The following environment variables shall affect the execution of *fold*:

92338 **LANG** Provide a default value for the internationalization variables that are unset or null.
 92339 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 92340 variables used to determine the values of locale categories.)

92341 **LC_ALL** If set to a non-empty string value, override the values of all the other
 92342 internationalization variables.

92343 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 92344 characters (for example, single-byte as opposed to multi-byte characters in
 92345 arguments and input files), and for the determination of the width in column
 92346 positions each character would occupy on a constant-width font output device.

92347 **LC_MESSAGES**

92348 Determine the locale that should be used to affect the format and contents of
 92349 diagnostic messages written to standard error.

92350 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

92351 **ASYNCHRONOUS EVENTS**

92352 Default.

92353 **STDOUT**

92354 The standard output shall be a file containing a sequence of characters whose order shall be
 92355 preserved from the input files, possibly with inserted <newline> characters.

92356 **STDERR**

92357 The standard error shall be used only for diagnostic messages.

92358 **OUTPUT FILES**

92359 None.

92360 **EXTENDED DESCRIPTION**

92361 None.

92362 **EXIT STATUS**

92363 The following exit values shall be returned:

92364 0 All input files were processed successfully.

92365 >0 An error occurred.

92366 **CONSEQUENCES OF ERRORS**

92367 Default.

92368 APPLICATION USAGE

92369 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.
92370 The *cut* utility should be used when the number of lines (or records) needs to remain constant.
92371 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

92372 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines
92373 wider than the printer is able to print (usually 80 or 132 column positions).

92374 EXAMPLES

92375 An example invocation that submits a file of possibly long lines to the printer (under the
92376 assumption that the user knows the line width of the printer to be assigned by *lp*):

```
92377 fold -w 132 bigfile | lp
```

92378 RATIONALE

92379 Although terminal input in canonical processing mode requires the erase character (frequently
92380 set to <backspace>) to erase the previous character (not byte or column position), terminal
92381 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the
92382 interpretation depends entirely on the physical device that actually displays/prints/stores the
92383 output. In all known internationalized implementations, the utilities producing output for
92384 mixed column-width output assume that a <backspace> character backs up one column position
92385 and outputs enough <backspace> characters to return to the start of the character when
92386 <backspace> is used to provide local line motions to support underlining and emboldening
92387 operations. Since *fold* without the **-b** option is dealing with these same constraints, <backspace>
92388 is always treated as backing up one column position rather than backing up one character.

92389 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column
92390 position when written out. This is no longer always true. Since the most common usage of *fold* is
92391 believed to be folding long lines for output to limited-length output devices, this capability was
92392 preserved as the default case. The **-b** option was added so that applications could *fold* files with
92393 arbitrary length lines into text files that could then be processed by the standard utilities. Note
92394 that although the width for the **-b** option is in bytes, a line is never split in the middle of a
92395 character. (It is unspecified what happens if a width is specified that is too small to hold a single
92396 character found in the input followed by a <newline>.)

92397 The tab stops are hardcoded to be every eighth column to meet historical practice. No new
92398 method of specifying other tab stops was invented.

92399 FUTURE DIRECTIONS

92400 None.

92401 SEE ALSO

92402 *cut*

92403 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

92404 CHANGE HISTORY

92405 First released in Issue 4.

92406 Issue 6

92407 The normative text is reworded to avoid use of the term “must” for application requirements.

92408 Issue 7

92409 Austin Group Interpretation 1003.1-2001 #092 is applied.

- 92410 Austin Group Interpretation 1003.1-2001 #204 is applied, updating the DESCRIPTION to clarify
- 92411 when a <newline> can be inserted before or after a <backspace>.
- 92412 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92413 **NAME**92414 fort77 — FORTRAN compiler (**FORTRAN**)92415 **SYNOPSIS**

```
92416 OB FD fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s]
92417 [-w] operand...
```

92418 **DESCRIPTION**

92419 The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full
 92420 FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually
 92421 consists of a compiler and link editor. The files referenced by *operands* are compiled and linked
 92422 to produce an executable file. It is unspecified whether the linking occurs entirely within the
 92423 operation of *fort77*; some implementations may produce objects that are not fully resolved until
 92424 the file is executed.

92425 If the `-c` option is present, for all pathname operands of the form *file.f*, the files:

92426 $\$(\text{basename } \textit{pathname.f}) .o$

92427 shall be created or overwritten as the result of successful compilation. If the `-c` option is not
 92428 specified, it is unspecified whether such `.o` files are created or deleted for the *file.f* operands.

92429 If there are no options that prevent link editing (such as `-c`) and all operands compile and link
 92430 without error, the resulting executable file shall be written into the file named by the `-o` option
 92431 (if present) or to the file **a.out**. The executable file shall be created as specified in the System
 92432 Interfaces volume of POSIX.1-2017, except that the file permissions shall be set to:

92433 `S_IRWXO | S_IRWXG | S_IRWXU`

92434 and that the bits specified by the *umask* of the process shall be cleared.

92435 **OPTIONS**

92436 The *fort77* utility shall conform to XBD [Section 12.2](#) (on page 216), except that:

92437 The `-I library` operands have the format of options, but their position within a list of
 92438 operands affects the order in which libraries are searched.

92439 The order of specifying the multiple `-L` options is significant.

92440 Conforming applications shall specify each option separately; that is, grouping option
 92441 letters (for example, `-cg`) need not be recognized by all implementations.

92442 The following options shall be supported:

92443 `-c` Suppress the link-edit phase of the compilation, and do not remove any object files
 92444 that are produced.

92445 `-g` Produce symbolic information in the object or executable files; the nature of this
 92446 information is unspecified, and may be modified by implementation-defined
 92447 interactions with other options.

92448 `-s` Produce object or executable files, or both, from which symbolic and other
 92449 information not required for proper execution using the *exec* family of functions
 92450 defined in the System Interfaces volume of POSIX.1-2017 has been removed
 92451 (stripped). If both `-g` and `-s` options are present, the action taken is unspecified.

92452 `-o outfile` Use the pathname *outfile*, instead of the default **a.out**, for the executable file
 92453 produced. If the `-o` option is present with `-c`, the result is unspecified.

92454 **-L** *directory* Change the algorithm of searching for the libraries named in **-I** operands to look in
 92455 the directory named by the *directory* pathname before looking in the usual places.
 92456 Directories named in **-L** options shall be searched in the specified order. At least
 92457 ten instances of this option shall be supported in a single *fort77* command
 92458 invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the
 92459 results are unspecified.

92460 **-O** *optlevel* Specify the level of code optimization. If the *optlevel* option-argument is the digit
 92461 '0', all special code optimizations shall be disabled. If it is the digit '1', the
 92462 nature of the optimization is unspecified. If the **-O** option is omitted, the nature of
 92463 the system's default optimization is unspecified. It is unspecified whether code
 92464 generated in the presence of the **-O 0** option is the same as that generated when
 92465 **-O** is omitted. Other *optlevel* values may be supported.

92466 **-w** Suppress warnings.

92467 Multiple instances of **-L** options can be specified.

92468 OPERANDS

92469 An *operand* is either in the form of a pathname or the form **-I library**. At least one operand of the
 92470 pathname form shall be specified. The following operands shall be supported:

92471 *file.f* The pathname of a FORTRAN source file to be compiled and optionally passed to
 92472 the link editor. The filename operand shall be of this form if the **-c** option is used.

92473 *file.a* A library of object files typically produced by *ar*, and passed directly to the link
 92474 editor. Implementations may recognize implementation-defined suffixes other
 92475 than **.a** as denoting object file libraries.

92476 *file.o* An object file produced by *fort77 -c* and passed directly to the link editor.
 92477 Implementations may recognize implementation-defined suffixes other than **.o** as
 92478 denoting object files.

92479 The processing of other files is implementation-defined.

92480 **-I library** (The letter ell.) Search the library named:

92481 *liblibrary.a*

92482 A library is searched when its name is encountered, so the placement of a **-I**
 92483 operand is significant. Several standard libraries can be specified in this manner, as
 92484 described in the EXTENDED DESCRIPTION section. Implementations may
 92485 recognize implementation-defined suffixes other than **.a** as denoting libraries.

92486 STDIN

92487 Not used.

92488 INPUT FILES

92489 The input file shall be one of the following: a text file containing FORTRAN source code; an
 92490 object file in the format produced by *fort77 -c*; or a library of object files, in the format produced
 92491 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
 92492 that produce files in these formats. Additional input files are implementation-defined.

92493 A <tab> encountered within the first six characters on a line of source code shall cause the
 92494 compiler to interpret the following character as if it were the seventh character on the line (that
 92495 is, in column 7).

92496 **ENVIRONMENT VARIABLES**92497 The following environment variables shall affect the execution of *fort77*:

92498 *LANG* Provide a default value for the internationalization variables that are unset or null.
 92499 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 92500 variables used to determine the values of locale categories.)

92501 *LC_ALL* If set to a non-empty string value, override the values of all the other
 92502 internationalization variables.

92503 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 92504 characters (for example, single-byte as opposed to multi-byte characters in
 92505 arguments and input files).

92506 *LC_MESSAGES*

92507 Determine the locale that should be used to affect the format and contents of
 92508 diagnostic messages written to standard error.

92509 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

92510 *TMPDIR* Determine the pathname that should override the default directory for temporary
 92511 files, if any.

92512 **ASYNCHRONOUS EVENTS**

92513 Default.

92514 **STDOUT**

92515 Not used.

92516 **STDERR**

92517 The standard error shall be used only for diagnostic messages. If more than one *file* operand
 92518 ending in *.f* (or possibly other unspecified suffixes) is given, for each such file:

92519 "%s:\n", <*file*>

92520 may be written to allow identification of the diagnostic message with the appropriate input file.

92521 This utility may produce warning messages about certain conditions that do not warrant
 92522 returning an error (non-zero) exit value.

92523 **OUTPUT FILES**

92524 Object files, listing files, and executable files shall be produced in unspecified formats.

92525 **EXTENDED DESCRIPTION**92526 **Standard Libraries**92527 The *fort77* utility shall recognize the following *-l* operand for the standard library:

92528 *-l f* This library contains all functions referenced in the ANSI X3.9-1978 standard. This
 92529 operand shall not be required to be present to cause a search of this library.

92530 In the absence of options that inhibit invocation of the link editor, such as *-c*, the *fort77* utility
 92531 shall cause the equivalent of a *-l f* operand to be passed to the link editor as the last *-l* operand,
 92532 causing it to be searched after all other object files and libraries are loaded.

92533 It is unspecified whether the library *libf.a* exists as a regular file. The implementation may
 92534 accept as *-l* operands names of objects that do not exist as regular files.

92535 **External Symbols**

92536 The FORTRAN compiler and link editor shall support the significance of external symbols up to
 92537 a length of at least 31 bytes; case folding is permitted. The action taken upon encountering
 92538 symbols exceeding the implementation-defined maximum symbol length is unspecified.

92539 The compiler and link editor shall support a minimum of 511 external symbols per source or
 92540 object file, and a minimum of 4095 external symbols total. A diagnostic message is written to
 92541 standard output if the implementation-defined limit is exceeded; other actions are unspecified.

92542 **EXIT STATUS**

92543 The following exit values shall be returned:

92544 0 Successful compilation or link edit.

92545 >0 An error occurred.

92546 **CONSEQUENCES OF ERRORS**

92547 When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and
 92548 continue to compile other source code operands. It shall return a non-zero exit status, but it is
 92549 implementation-defined whether an object module is created. If the link edit is unsuccessful, a
 92550 diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero status.

92551 **APPLICATION USAGE**

92552 None.

92553 **EXAMPLES**

92554 The following usage example compiles **xyz.f** and creates the executable file **foo**:

```
92555 fort77 -o foo xyz.f
```

92556 The following example compiles **xyz.f** and creates the object file **xyz.o**:

```
92557 fort77 -c xyz.f
```

92558 The following example compiles **xyz.f** and creates the executable file **a.out**:

```
92559 fort77 xyz.f
```

92560 The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

```
92561 fort77 xyz.f b.o
```

92562 **RATIONALE**

92563 The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The
 92564 name *f77* was not chosen to avoid problems with historical implementations. The
 92565 ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of
 92566 FORTRAN-77 has been superseded by the ISO/IEC 1539:1991 standard.

92567 The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not
 92568 included in this volume of POSIX.1-2017 ¶even though they are commonly implemented ¶since
 92569 there is no requirement that the FORTRAN compiler use the C preprocessor.

92570 The **-onetrip** option was not included in this volume of POSIX.1-2017, even though many
 92571 historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism
 92572 that should not be perpetuated.

92573 Some implementations produce compilation listings. This aspect of FORTRAN has been left
 92574 unspecified because there was controversy concerning the various methods proposed for
 92575 implementing it: a **-V** option overlapped with historical vendor practice and a naming
 92576 convention of creating files with **.I** suffixes collided with historical *lex* file naming practice.

92577 There is no `-I` option in this version of this volume of POSIX.1-2017 to specify a directory for file
 92578 inclusion. An `INCLUDE` directive has been a part of the Fortran-90 discussions, but an interface
 92579 supporting that standard is not in the current scope.

92580 It is noted that many FORTRAN compilers produce an object module even when compilation
 92581 errors occur; during a subsequent compilation, the compiler may patch the object module rather
 92582 than recompiling all the code. Consequently, it is left to the implementor whether or not an
 92583 object file is created.

92584 A reference to MIL-STD-1753 was removed from an early proposal in response to a request from
 92585 the POSIX FORTRAN-binding standard developers. It was not the intention of the standard
 92586 developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not
 92587 specify the military standard or any special preprocessing requirements. Furthermore, use of
 92588 that document would have been inappropriate for an international standard.

92589 The specification of optimization has been subject to changes through early proposals. At one
 92590 time, `-O` and `-N` were Booleans: optimize and do not optimize (with an unspecified default).
 92591 Some historical practice led this to be changed to:

- 92592 `-O 0` No optimization.
- 92593 `-O 1` Some level of optimization.
- 92594 `-O n` Other, unspecified levels of optimization.

92595 It is not always clear whether “good code generation” is the same thing as optimization. Simple
 92596 optimizations of local actions do not usually affect the semantics of a program. The `-O 0` option
 92597 has been included to accommodate the very particular nature of scientific calculations in a
 92598 highly optimized environment; compilers make errors. Some degree of optimization is expected,
 92599 even if it is not documented here, and the ability to shut it off completely could be important
 92600 when porting an application. An implementation may treat `-O 0` as “do less than normal” if it
 92601 wishes, but this is only meaningful if any of the operations it performs can affect the semantics
 92602 of a program. It is highly dependent on the implementation whether doing less than normal is
 92603 logical. It is not the intent of the `-O 0` option to ask for inefficient code generation, but rather to
 92604 assure that any semantically visible optimization is suppressed.

92605 The specification of standard library access is consistent with the C compiler specification.
 92606 Implementations are not required to have `/usr/lib/libf.a`, as many historical implementations do,
 92607 but if not they are required to recognize `f` as a token.

92608 External symbol size limits are in normative text; conforming applications need to know these
 92609 limits. However, the minimum maximum symbol length should be taken as a constraint on a
 92610 conforming application, not on an implementation, and consequently the action taken for a
 92611 symbol exceeding the limit is unspecified. The minimum size for the external symbol table was
 92612 added for similar reasons.

92613 The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when
 92614 compilation or link-edit errors occur. The behavior of several historical implementations was
 92615 examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in
 92616 the face of compiler or linker errors. If a linker writes the executable file, then links it on disk
 92617 with `lseek()`s and `write()`s, the partially linked executable file can be left on disk and its execute
 92618 bits turned off if the link edit fails. However, if the linker links the image in memory before
 92619 writing the file to disk, it need not touch the executable file (if it already exists) because the link
 92620 edit fails. Since both approaches are historical practice, a conforming application shall rely on
 92621 the exit status of `fort77`, rather than on the existence or mode of the executable file.

92622 The `-g` and `-s` options are not specified as mutually-exclusive. Historically, these two options

92623 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate
92624 to leave their interaction unspecified.

92625 The requirement that conforming applications specify compiler options separately is to reserve
92626 the multi-character option name space for vendor-specific compiler options, which are known to
92627 exist in many historical implementations. Implementations are not required to recognize, for
92628 example, `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The SYNOPSIS shows all
92629 of the options separately to highlight this requirement on applications.

92630 Echoing filenames to standard error is considered a diagnostic message because it would
92631 otherwise be difficult to associate an error message with the erring file. They are described with
92632 “may” to allow implementations to use other methods of identifying files and to parallel the
92633 description in *c99*.

92634 **FUTURE DIRECTIONS**

92635 Future versions of this standard may withdraw this utility. There are implementations of
92636 compilers that conform to much more recent versions of the FORTRAN programming language.
92637 Since there is no active FORTRAN binding to POSIX.1-2017, this standard does not need to
92638 specify any compiler.

92639 **SEE ALSO**

92640 *ar*, *asa*, *c99*, *umask*

92641 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

92642 XSH *exec*

92643 **CHANGE HISTORY**

92644 First released in Issue 4.

92645 **Issue 6**

92646 This utility is marked as part of the FORTRAN Development Utilities option.

92647 The normative text is reworded to avoid use of the term “must” for application requirements.

92648 **Issue 7**

92649 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92650 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0102 [546] and XCU/TC2-2008/0103
92651 [546] are applied.

92652 **NAME**

92653 fuser — list process IDs of all processes that have one or more files open

92654 **SYNOPSIS**92655 XSI `fuser [-cfu] file...`92656 **DESCRIPTION**92657 The *fuser* utility shall write to standard output the process IDs of processes running on the local
92658 system that have one or more named files open. For block special devices, all processes using
92659 any file on that device are listed.92660 The *fuser* utility shall write to standard error additional information about the named files
92661 indicating how the file is being used.

92662 Any output for processes running on remote systems that have a named file open is unspecified.

92663 A user may need appropriate privileges to invoke the *fuser* utility.92664 **OPTIONS**92665 The *fuser* utility shall conform to XBD [Section 12.2](#) (on page 216).

92666 The following options shall be supported:

92667 **-c** The file is treated as a mount point and the utility shall report on any files open in
92668 the file system.92669 **-f** The report shall be only for the named files.92670 **-u** The user name, in parentheses, associated with each process ID written to standard
92671 output shall be written to standard error.92672 **OPERANDS**

92673 The following operand shall be supported:

92674 *file* A pathname on which the file or file system is to be reported.92675 **STDIN**

92676 Not used.

92677 **INPUT FILES**

92678 The user database.

92679 **ENVIRONMENT VARIABLES**92680 The following environment variables shall affect the execution of *fuser*:92681 **LANG** Provide a default value for the internationalization variables that are unset or null.
92682 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
92683 variables used to determine the values of locale categories.)92684 **LC_ALL** If set to a non-empty string value, override the values of all the other
92685 internationalization variables.92686 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
92687 characters (for example, single-byte as opposed to multi-byte characters in
92688 arguments).92689 **LC_MESSAGES**92690 Determine the locale that should be used to affect the format and contents of
92691 diagnostic messages written to standard error.

- 92692 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 92693 **ASYNCHRONOUS EVENTS**
- 92694 Default.
- 92695 **STDOUT**
- 92696 The *fuser* utility shall write the process ID for each process using each file given as an operand to
92697 standard output in the following format:
- 92698 "%d", <*process_id*>
- 92699 **STDERR**
- 92700 The *fuser* utility shall write diagnostic messages to standard error.
- 92701 The *fuser* utility also shall write the following to standard error:
- 92702 The pathname of each named file is written followed immediately by a <colon>.
- 92703 For each process ID written to standard output, the character 'c' shall be written to
92704 standard error if the process is using the file as its current directory and the character 'r'
92705 shall be written to standard error if the process is using the file as its root directory.
92706 Implementations may write other alphabetic characters to indicate other uses of files.
- 92707 When the *-u* option is specified, characters indicating the use of the file shall be followed
92708 immediately by the user name, in parentheses, corresponding to the real user ID of the
92709 process. If the user name cannot be resolved from the real user ID of the process, the real
92710 user ID of the process shall be written instead of the user name.
- 92711 When standard output and standard error are directed to the same file, the output shall be
92712 interleaved so that the filename appears at the start of each line, followed by the process ID and
92713 characters indicating the use of the file. Then, if the *-u* option is specified, the user name or user
92714 ID for each process using that file shall be written.
- 92715 A <newline> shall be written to standard error after the last output described above for each *file*
92716 operand.
- 92717 **OUTPUT FILES**
- 92718 None.
- 92719 **EXTENDED DESCRIPTION**
- 92720 None.
- 92721 **EXIT STATUS**
- 92722 The following exit values shall be returned:
- 92723 0 Successful completion.
- 92724 >0 An error occurred.
- 92725 **CONSEQUENCES OF ERRORS**
- 92726 Default.

92727 **APPLICATION USAGE**

92728 None.

92729 **EXAMPLES**

92730 The command:

92731 `fuser -fu .`

92732 writes to standard output the process IDs of processes that are using the current directory and
92733 writes to standard error an indication of how those processes are using the directory and the
92734 user names associated with the processes that are using the current directory.

92735 `fuser -c <mount point>`

92736 writes to standard output the process IDs of processes that are using any file in the file system
92737 which is mounted on <mount point> and writes to standard error an indication of how those
92738 processes are using the files.

92739 `fuser <mount point>`

92740 writes to standard output the process IDs of processes that are using the file which is named by
92741 <mount point> and writes to standard error an indication of how those processes are using the
92742 file.

92743 `fuser <block device>`

92744 writes to standard output the process IDs of processes that are using any file which is on the
92745 device named by <block device> and writes to standard error an indication of how those
92746 processes are using the file.

92747 `fuser -f <block device>`

92748 writes to standard output the process IDs of processes that are using the file <block device> itself
92749 and writes to standard error an indication of how those processes are using the file.

92750 **RATIONALE**92751 The definition of the *fuser* utility follows existing practice.92752 **FUTURE DIRECTIONS**

92753 None.

92754 **SEE ALSO**92755 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)92756 **CHANGE HISTORY**

92757 First released in Issue 5.

92758 **Issue 7**

92759 SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

92760 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

92761 **NAME**

92762 gencat ‡generate a formatted message catalog

92763 **SYNOPSIS**92764 gencat *catfile* *msgfile*...92765 **DESCRIPTION**

92766 The *gencat* utility shall merge the message text source file *msgfile* into a formatted message
 92767 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its
 92768 messages shall be included in the new *catfile*. If set and message numbers collide, the new
 92769 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

92770 **OPTIONS**

92771 None.

92772 **OPERANDS**

92773 The following operands shall be supported:

92774 *catfile* A pathname of the formatted message catalog. If '-' is specified, standard output
 92775 shall be used. The format of the message catalog produced is unspecified.

92776 *msgfile* A pathname of a message text source file. If '-' is specified for an instance of
 92777 *msgfile*, standard input shall be used. The format of message text source files is
 92778 defined in the EXTENDED DESCRIPTION section.

92779 **STDIN**92780 The standard input shall not be used unless a *msgfile* operand is specified as '-'.92781 **INPUT FILES**

92782 The input files shall be text files.

92783 **ENVIRONMENT VARIABLES**92784 The following environment variables shall affect the execution of *gencat*:

92785 *LANG* Provide a default value for the internationalization variables that are unset or null.
 92786 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 92787 variables used to determine the values of locale categories.)

92788 *LC_ALL* If set to a non-empty string value, override the values of all the other
 92789 internationalization variables.

92790 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 92791 characters (for example, single-byte as opposed to multi-byte characters in
 92792 arguments and input files).

92793 *LC_MESSAGES*

92794 Determine the locale that should be used to affect the format and contents of
 92795 diagnostic messages written to standard error.

92796 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

92797 **ASYNCHRONOUS EVENTS**

92798 Default.

92799 **STDOUT**92800 The standard output shall not be used unless the *catfile* operand is specified as '-'.

92801 **STDERR**

92802 The standard error shall be used only for diagnostic messages.

92803 **OUTPUT FILES**

92804 None.

92805 **EXTENDED DESCRIPTION**92806 The content of a message text file shall be in the format defined as follows. Note that the fields of
92807 a message text source line are separated by a single <blank> character. Any other <blank>
92808 characters are considered to be part of the subsequent field.92809 **\$set** *n comment*92810 This line specifies the set identifier of the following messages until the next **\$set** or
92811 end-of-file appears. The *n* denotes the set identifier, which is defined as a number
92812 in the range [1, {NL_SETMAX}] (see the <limits.h> header defined in the Base
92813 Definitions volume of POSIX.1-2017). The application shall ensure that set
92814 identifiers are presented in ascending order within a single source file, but need
92815 not be contiguous. Any string following the set identifier shall be treated as a
92816 comment. If no **\$set** directive is specified in a message text source file, all messages
92817 shall be located in an implementation-defined default message set NL_SETD (see
92818 the <nl_types.h> header defined in the Base Definitions volume of POSIX.1-2017).92819 **\$delset** *n comment*92820 This line deletes message set *n* from an existing message catalog. The *n* denotes the
92821 set number [1, {NL_SETMAX}]. Any string following the set number shall be
92822 treated as a comment.92823 **\$ comment** A line beginning with '\$' followed by a <blank> shall be treated as a comment.92824 *m message-text*92825 The *m* denotes the message identifier, which is defined as a number in the range [1,
92826 {NL_MSGMAX}] (see the <limits.h> header). The *message-text* shall be stored in the
92827 message catalog with the set identifier specified by the last **\$set** directive, and with
92828 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is
92829 present, an empty string shall be stored in the message catalog. If a message source
92830 line has a message number, but neither a field separator nor *message-text*, the
92831 existing message with that number (if any) shall be deleted from the catalog. The
92832 application shall ensure that message identifiers are in ascending order within a
92833 single set, but need not be contiguous. The application shall ensure that the length
92834 of *message-text* is in the range [0, {NL_TEXTMAX}] (see the <limits.h> header).92835 **\$quote** *n* This line specifies an optional quote character *c*, which can be used to surround
92836 *message-text* so that trailing <space> characters or null (empty) messages are visible
92837 in a message source line. By default, or if an empty **\$quote** directive is supplied, no
92838 quoting of *message-text* shall be recognized.92839 Empty lines in a message text source file shall be ignored. The effects of lines starting with any
92840 character other than those defined above are implementation-defined.92841 Text strings can contain the special characters and escape sequences defined in the following
92842 table:

	Description	Symbol	Sequence
92843	<newline>	NL(LF)	\n
92844	Horizontal-tab	HT	\t
92845	<vertical-tab>	VT	\v
92846	<backspace>	BS	\b
92847	<carriage-return>	CR	\r
92848	<form-feed>	FF	\f
92849	Backslash	\	\\
92850	Bit pattern	ddd	\\ddd

92852 The escape sequence "\ddd" consists of <backslash> followed by one, two, or three octal digits,
 92853 which shall be taken to specify the value of the desired character. If the character following a
 92854 <backslash> is not one of those specified, the <backslash> shall be ignored.

92855 A <backslash> followed by a <newline> is also used to continue a string on the following line.
 92856 Thus, the following two lines describe a single message string:

```
92857 1 This line continues \  

  92858 to the next line
```

92859 which shall be equivalent to:

```
92860 1 This line continues to the next line
```

92861 EXIT STATUS

92862 The following exit values shall be returned:

92863 0 Successful completion.

92864 >0 An error occurred.

92865 CONSEQUENCES OF ERRORS

92866 Default.

92867 APPLICATION USAGE

92868 Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot
 92869 be guaranteed between different types of machine. Thus, just as C programs need to be
 92870 recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

92871 EXAMPLES

92872 None.

92873 RATIONALE

92874 None.

92875 FUTURE DIRECTIONS

92876 None.

92877 SEE ALSO

92878 [iconv](#)

92879 XBD [Chapter 8](#) (on page 173), [<limits.h>](#), [<nl_types.h>](#)

92880 CHANGE HISTORY

92881 First released in Issue 3.

92882 Issue 6

92883 The normative text is reworded to avoid use of the term “must” for application requirements.

92884 **Issue 7**
92885

The *gencat* utility is moved from the XSI option to the Base.

92886 **NAME**92887 `get` ‡get a version of an SCCS file **(DEVELOPMENT)**92888 **SYNOPSIS**92889 `get [-begkmnlLpst] [-c cutoff] [-i list] [-r SID] [-x list] file...`92890 **DESCRIPTION**92891 The *get* utility shall generate a text file from each named SCCS *file* according to the specifications
92892 given by its options.92893 The generated text shall normally be written into a file called the **g-file** whose name is derived
92894 from the SCCS filename by simply removing the leading "s.". 92895 **OPTIONS**92896 The *get* utility shall conform to XBD [Section 12.2](#) (on page 216).

92897 The following options shall be supported:

92898 **-r** *SID* Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file
92899 to be retrieved. The table shows, for the most useful cases, what version of an
92900 SCCS file is retrieved (as well as the SID of the version to be eventually created by
92901 *delta* if the **-e** option is also used), as a function of the SID specified.92902 **-c** *cutoff* Indicate the *cutoff* date-time, in the form:92903 `YY[MM[DD[HH[MM[SS]]]]]`92904 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999
92905 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.92906 **Note:** It is expected that in a future version of this standard the default century inferred
92907 from a 2-digit year will change. (This would apply to all commands accepting a
92908 2-digit year as input.)92909 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
92910 date-time shall be included in the generated text file. Units omitted from the date-
92911 time default to their maximum possible values; for example, **-c** 7502 is equivalent
92912 to **-c** 750228235959.92913 Any number of non-numeric characters may separate the various 2-digit pieces of
92914 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:
92915 **-c** "77/2/2 9:22:25".92916 **-e** Indicate that the *get* is for the purpose of editing or making a change (delta) to the
92917 SCCS file via a subsequent use of *delta*. The **-e** option used in a *get* for a particular
92918 version (SID) of the SCCS file shall prevent further *get* commands from editing on
92919 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.
92920 Concurrent use of *get* **-e** for different SIDs is always allowed.92921 If the **g-file** generated by *get* with a **-e** option is accidentally ruined in the process
92922 of editing, it may be regenerated by re-executing the *get* command with the **-k**
92923 option in place of the **-e** option.92924 SCCS file protection specified via the ceiling, floor, and authorized user list stored
92925 in the SCCS file shall be enforced when the **-e** option is used.92926 **-b** Use with the **-e** option to indicate that the new delta should have an SID in a new
92927 branch as shown in the table below. This option shall be ignored if the **b** flag is not
92928 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that
92929 has no successors on the SCCS file tree.)

92930		Note: A branch delta may always be created from a non-leaf delta.
92931	-i list	Indicate a <i>list</i> of deltas to be included (forced to be applied) in the creation of the generated file. The <i>list</i> has the following syntax:
92932		
92933		<code><list> ::= <range> <list> , <range></code>
92934		<code><range> ::= SID SID - SID</code>
92935		SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of the table in the EXTENDED DESCRIPTION section, except that the result of supplying a partial SID is unspecified. A diagnostic message shall be written if the first SID in the range is not an ancestor of the second SID in the range.
92936		
92937		
92938		
92939		
92940	-x list	Indicate a <i>list</i> of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the -i option for the <i>list</i> format.
92941		
92942	-k	Suppress replacement of identification keywords (see below) in the retrieved text by their value. The -k option shall be implied by the -e option.
92943		
92944	-l	Write a delta summary into an l-file .
92945	-L	Write a delta summary to standard output. All informative output that normally is written to standard output shall be written to standard error instead, unless the -s option is used, in which case it shall be suppressed.
92946		
92947		
92948	-p	Write the text retrieved from the SCCS file to the standard output. No g-file shall be created. All informative output that normally goes to the standard output shall go to standard error instead, unless the -s option is used, in which case it shall disappear.
92949		
92950		
92951		
92952	-s	Suppress all informative output normally written to standard output. However, fatal error messages (which shall always be written to the standard error) shall remain unaffected.
92953		
92954		
92955	-m	Precede each text line retrieved from the SCCS file by the SID of the delta that inserted the text line in the SCCS file. The format shall be:
92956		
92957		<code>"%s\t%s", <SID>, <text line></code>
92958	-n	Precede each generated text line with the %M% identification keyword value (see below). The format shall be:
92959		
92960		<code>"%s\t%s", <%M% value>, <text line></code>
92961		When both the -m and -n options are used, the <code><text line></code> shall be replaced by the -m option-generated format.
92962		
92963	-g	Suppress the actual retrieval of text from the SCCS file. It is primarily used to generate an l-file , or to verify the existence of a particular SID.
92964		
92965	-t	Use to access the most recently created (top) delta in a given release (for example, -r 1), or release and level (for example, -r 1.2).
92966		

92967 **OPERANDS**

92968 The following operands shall be supported:

92969 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*
 92970 utility shall behave as though each file in the directory were specified as a named
 92971 file, except that non-SCCS files (last component of the pathname does not begin
 92972 with **s**.) and unreadable files shall be silently ignored.

92973 If exactly one *file* operand appears, and it is **'-'**, the standard input shall be read;
 92974 each line of the standard input is taken to be the name of an SCCS file to be
 92975 processed. Non-SCCS files and unreadable files shall be silently ignored.

92976 **STDIN**

92977 The standard input shall be a text file used only if the *file* operand is specified as **'-'**. Each line
 92978 of the text file shall be interpreted as an SCCS pathname.

92979 **INPUT FILES**

92980 The SCCS files shall be files of an unspecified format.

92981 **ENVIRONMENT VARIABLES**92982 The following environment variables shall affect the execution of *get*:

92983 *LANG* Provide a default value for the internationalization variables that are unset or null.
 92984 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 92985 variables used to determine the values of locale categories.)

92986 *LC_ALL* If set to a non-empty string value, override the values of all the other
 92987 internationalization variables.

92988 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 92989 characters (for example, single-byte as opposed to multi-byte characters in
 92990 arguments and input files).

92991 *LC_MESSAGES*

92992 Determine the locale that should be used to affect the format and contents of
 92993 diagnostic messages written to standard error, and informative messages written
 92994 to standard output (or standard error, if the **-p** option is used).

92995 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

92996 *TZ* Determine the timezone in which the times and dates written in the SCCS file are
 92997 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default
 92998 timezone is used.

92999 **ASYNCHRONOUS EVENTS**

93000 Default.

93001 **STDOUT**

93002 For each file processed, *get* shall write to standard output the SID being accessed and the
 93003 number of lines retrieved from the SCCS file, in the following format:

93004 "%s\n%d lines\n", <SID>, <number of lines>

93005 If the **-e** option is used, the SID of the delta to be made shall appear after the SID accessed and
 93006 before the number of lines generated, in the POSIX locale:

93007 "%s\nnew delta %s\n%d lines\n", <SID accessed>,
 93008 <SID to be made>, <number of lines>

93009 If there is more than one named file or if a directory or standard input is named, each pathname

93010 shall be written before each of the lines shown in one of the preceding formats:

93011 `"\n%s:\n", <pathname>`

93012 If the `-L` option is used, a delta summary shall be written following the format specified below
93013 for **l-files**.

93014 If the `-i` option is used, included deltas shall be listed following the notation, in the POSIX
93015 locale:

93016 `"Included:\n"`

93017 If the `-x` option is used, excluded deltas shall be listed following the notation, in the POSIX
93018 locale:

93019 `"Excluded:\n"`

93020 If the `-p` or `-L` options are specified, the standard output shall consist of the text retrieved from
93021 the SCCS file.

93022 **STDERR**

93023 The standard error shall be used only for diagnostic messages, except if the `-p` or `-L` options are
93024 specified, it shall include all informative messages normally sent to standard output.

93025 **OUTPUT FILES**

93026 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-**
93027 **file**, **p-file**, and **z-file**. The letter before the <hyphen-minus> is called the *tag*. An auxiliary
93028 filename shall be formed from the SCCS filename: the application shall ensure that the last
93029 component of all SCCS filenames is of the form *s.module-name*; the auxiliary files shall be named
93030 by replacing the leading *s* with the tag. The **g-file** shall be an exception to this scheme: the **g-file**
93031 is named by removing the *s*. prefix. For example, for *s.xyz.c*, the auxiliary filenames would be
93032 **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

93033 The **g-file**, which contains the generated text, shall be created in the current directory (unless the
93034 `-p` option is used). A **g-file** shall be created in all cases, whether or not any lines of text were
93035 generated by the *get*. It shall be owned by the real user. If the `-k` option is used or implied, the
93036 **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be
93037 read-only. Only the real user need have write permission in the current directory.

93038 The **l-file** shall contain a table showing which deltas were applied in generating the retrieved
93039 text. The **l-file** shall be created in the current directory if the `-l` option is used; it shall be read-
93040 only and it is owned by the real user. Only the real user need have write permission in the
93041 current directory.

93042 Lines in the **l-file** shall have the following format:

93043 `"%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,
93044 <SID>, <date-time>, <login>`

93045 where the entries are:

93046 `<code1>` A <space> if the delta was applied; ' * ' otherwise.

93047 `<code2>` A <space> if the delta was applied or was not applied and ignored; ' * ' if the delta
93048 was not applied and was not ignored.

93049 `<code3>` A character indicating a special reason why the delta was or was not applied:

93050 **I** Included.

93051 X Excluded.

93052 C Cut off (by a `-c` option).

93053 <date-time> Date and time (using the format of the *date* utility's `%Y/%m/%d %T` conversion
93054 specification format) of creation.

93055 <login> Login name of person who created *delta*.

93056 The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line
93057 shall terminate each entry.

93058 The **p-file** shall be used to pass information resulting from a *get* with a `-e` option along to *delta*.
93059 Its contents shall also be used to prevent a subsequent execution of *get* with a `-e` option for the
93060 same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be
93061 created in the directory containing the SCCS file and the application shall ensure that the
93062 effective user has write permission in that directory. It shall be writable by owner only, and
93063 owned by the effective user. Each line in the **p-file** shall have the following format:

93064 "%sΔ%sΔ%sΔ%s%s\n", <g-file SID>,
93065 <SID of new delta>, <login-name of real user>,
93066 <date-time>, <i-value>, <x-value>

93067 where <i-value> uses the format " " if no `-i` option was specified, and shall use the format:

93068 "Δ-i%s", <-i option option-argument>

93069 if a `-i` option was specified and <x-value> uses the format " " if no `-x` option was specified, and
93070 shall use the format:

93071 "Δ-x%s", <-x option option-argument>

93072 if a `-x` option was specified. There can be an arbitrary number of lines in the **p-file** at any time;
93073 no two lines shall have the same new delta SID.

93074 The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall
93075 be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created
93076 in the directory containing the SCCS file for the duration of *get*. The same protection restrictions
93077 as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.

93078 EXTENDED DESCRIPTION

93079

Determination of SCCS Identification String					
SID* Specified	-b Keyletter Used	Other Conditions	SID Retrieved	SID of Delta to be Created	
93080	none	no	R defaults to mR	mR.mL	mR.(mL+1)
93081	none	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
93082	R	no	R > mR	mR.mL	R.1***
93083	R	no	R = mR	mR.mL	mR.(mL+1)
93084	R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
93085	R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
93086	R	-	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
93087	R	-	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
93088	R.L	no	No trunk successor	R.L	R.(L+1)
93089	R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
93090	R.L	-	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
93091	R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
93092	R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
93093	R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
93094	R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
93095	R.L.B.S	-	Branch successor	R.L.B.S	R.L.(mB+1).1

93101 * R, L, B, and S are the release, level, branch, and sequence components of the SID,
 93102 respectively; m means maximum. Thus, for example, R.mL means “the maximum level
 93103 number within release R”; R.L.(mB+1).1 means “the first sequence number on the new
 93104 branch (that is, maximum branch number plus one) of level L within release R”. Note
 93105 that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified
 93106 components shall exist.

93107 ** hR is the highest existing release that is lower than the specified, nonexistent, release R.

93108 *** This is used to force creation of the first delta in a new release.

93109 The -b option is effective only if the b flag is present in the file. An entry of ‘-’ means
 93110 “irrelevant”.

93111 This case applies if the d (default SID) flag is not present in the file. If the d flag is
 93112 present in the file, then the SID obtained from the d flag is interpreted as if it had been
 93113 specified on the command line. Thus, one of the other cases in this table applies.

93114 **System Date and Time**

93115 When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into
 93116 account. If any of these times are apparently in the future, the behavior is unspecified.

93117 **Identification Keywords**

93118 Identifying information shall be inserted into the text retrieved from the SCCS file by replacing
 93119 identification keywords with their value wherever they occur. The following keywords may be
 93120 used in the text stored in an SCCS file:

93121 **%M%** Module name: either the value of the **m** flag in the file, or if absent, the name of the
 93122 SCCS file with the leading **s.** removed.

93123 **%I%** SCCS identification (SID) (**%R%.%L%** or **%R%.%L%.%B%.%S%**) of the retrieved
 93124 text.

93125 **%R%** Release.

93126 **%L%** Level.

93127 **%B%** Branch.

93128 **%S%** Sequence.

93129 **%D%** Current date (*YY/MM/DD*).

93130 **%H%** Current date (*MM/DD/YY*).

93131 **%T%** Current time (*HH:MM:SS*).

93132 **%E%** Date newest applied delta was created (*YY/MM/DD*).

93133 **%G%** Date newest applied delta was created (*MM/DD/YY*).

93134 **%U%** Time newest applied delta was created (*HH:MM:SS*).

93135 **%Y%** Module type: value of the **t** flag in the SCCS file.

93136 **%F%** SCCS filename.

93137 **%P%** SCCS absolute pathname.

93138 **%Q%** The value of the **q** flag in the file.

93139 **%C%** Current line number. This keyword is intended for identifying messages output by
 93140 the program, such as "this should not have happened" type errors. It is not
 93141 intended to be used on every line to provide sequence numbers.

93142 **%Z%** The four-character string "**@ (#)**" recognizable by *what*.

93143 **%W%** A shorthand notation for constructing *what* strings:

93144 `%W%=%Z%%M%<tab>%I%`

93145 **%A%** Another shorthand notation for constructing *what* strings:

93146 `%A%=%Z%%Y%%M%%I%%Z%`

93147 **EXIT STATUS**

93148 The following exit values shall be returned:

93149 0 Successful completion.

93150 >0 An error occurred.

93151 CONSEQUENCES OF ERRORS

93152 Default.

93153 APPLICATION USAGE

93154 Problems can arise if the system date and time have been modified (for example, put forward
93155 and then back again, or unsynchronized clocks across a network) and can also arise when
93156 different values of the *TZ* environment variable are used.

93157 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares
93158 the previous file body against the working file as part of its normal operation.

93159 EXAMPLES

93160 None.

93161 RATIONALE

93162 None.

93163 FUTURE DIRECTIONS

93164 None.

93165 SEE ALSO

93166 *admin, delta, prs, what*

93167 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

93168 CHANGE HISTORY

93169 First released in Issue 2.

93170 Issue 5

93171 A correction is made to the first format string in STDOUT.

93172 The interpretation of the *YY* component of the *-c cutoff* argument is noted.

93173 Issue 6

93174 The obsolescent SYNOPSIS is removed, removing the *-lp* option.

93175 The normative text is reworded to avoid use of the term “must” for application requirements.

93176 The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

93177 The Open Group Corrigendum U048/1 is applied.

93178 The Open Group Interpretation PIN4C.00014 is applied.

93179 The Open Group Base Resolution bwg2001-007 is applied as follows:

93180 The EXTENDED DESCRIPTION section is updated to make partial SID handling
93181 unspecified, reflecting common usage, and to clarify SID ranges.

93182 New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections
93183 regarding how the system date and time may be taken into account.

93184 The *TZ* environment variable is added to the ENVIRONMENT VARIABLES section.

93185 Issue 7

93186 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93187 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0104 [584] is applied.

93188 **NAME**93189 `getconf` \ddagger get configuration values93190 **SYNOPSIS**93191 `getconf` [-v *specification*] *system_var*93192 `getconf` [-v *specification*] *path_var pathname*93193 **DESCRIPTION**93194 In the first synopsis form, the *getconf* utility shall write to the standard output the value of the
93195 variable specified by the *system_var* operand.93196 In the second synopsis form, the *getconf* utility shall write to the standard output the value of the
93197 variable specified by the *path_var* operand for the path specified by the *pathname* operand.93198 The value of each configuration variable shall be determined as if it were obtained by calling the
93199 function from which it is defined to be available by this volume of POSIX.1-2017 or by the
93200 System Interfaces volume of POSIX.1-2017 (see the OPERANDS section). The value shall reflect
93201 conditions in the current operating environment.93202 **OPTIONS**93203 The *getconf* utility shall conform to XBD [Section 12.2](#) (on page 216).

93204 The following option shall be supported:

93205 **-v** *specification*93206 Indicate a specific specification and version for which configuration variables shall
93207 be determined. If this option is not specified, the values returned correspond to an
93208 implementation default conforming compilation environment.

93209 If the command:

93210 `getconf _POSIX_V7_ILP32_OFF32`93211 does not write "-1\n" or "undefined\n" to standard output, then commands of
93212 the form:93213 `getconf -v POSIX_V7_ILP32_OFF32 ...`93214 determine values for configuration variables corresponding to the
93215 POSIX_V7_ILP32_OFF32 compilation environment specified in [c99](#), the
93216 EXTENDED DESCRIPTION.

93217 If the command:

93218 `getconf _POSIX_V7_ILP32_OFFBIG`93219 does not write "-1\n" or "undefined\n" to standard output, then commands of
93220 the form:93221 `getconf -v POSIX_V7_ILP32_OFFBIG ...`93222 determine values for configuration variables corresponding to the
93223 POSIX_V7_ILP32_OFFBIG compilation environment specified in [c99](#), the
93224 EXTENDED DESCRIPTION.

93225 If the command:

93226 `getconf _POSIX_V7_LP64_OFF64`93227 does not write "-1\n" or "undefined\n" to standard output, then commands of
93228 the form:

93229 `getconf -v POSIX_V7_LP64_OFF64 ...`
 93230 determine values for configuration variables corresponding to the
 93231 POSIX_V7_LP64_OFF64 compilation environment specified in [c99](#), the
 93232 EXTENDED DESCRIPTION.

93233 If the command:

93234 `getconf _POSIX_V7_LPBIG_OFFBIG`

93235 does not write "-1\n" or "undefined\n" to standard output, then commands of
 93236 the form:

93237 `getconf -v POSIX_V7_LPBIG_OFFBIG ...`

93238 determine values for configuration variables corresponding to the
 93239 POSIX_V7_LPBIG_OFFBIG compilation environment specified in [c99](#), the
 93240 EXTENDED DESCRIPTION.

93241 OPERANDS

93242 The following operands shall be supported:

93243 *path_var* A name of a configuration variable. All of the variables in the Variable column of
 93244 the table in the DESCRIPTION of the *fpathconf()* function defined in the System
 93245 Interfaces volume of POSIX.1-2017, without the enclosing braces, shall be
 93246 supported. The implementation may add other local variables.

93247 *pathname* A pathname for which the variable specified by *path_var* is to be determined.

93248 *system_var* A name of a configuration variable. All of the following variables shall be
 93249 supported:

93250 The names in the Variable column of the table in the DESCRIPTION of the
 93251 *sysconf()* function in the System Interfaces volume of POSIX.1-2017, except
 93252 for the entries corresponding to `_SC_CLK_TCK`, `_SC_GETGR_R_SIZE_MAX`,
 93253 and `_SC_GETPW_R_SIZE_MAX`, without the enclosing braces.

93254 For compatibility with earlier versions, the following variable names shall
 93255 also be supported:

93256 `POSIX2_C_BIND`
 93257 `POSIX2_C_DEV`
 93258 `POSIX2_CHAR_TERM`
 93259 `POSIX2_FORT_DEV`
 93260 `POSIX2_FORT_RUN`
 93261 `POSIX2_LOCALEDEF`
 93262 `POSIX2_SW_DEV`
 93263 `POSIX2_UPE`
 93264 `POSIX2_VERSION`

93265 and shall be equivalent to the same name prefixed with an <underscore>.
 93266 This requirement may be removed in a future version.

93267 The names of the symbolic constants used as the *name* argument of the
 93268 *confstr()* function in the System Interfaces volume of POSIX.1-2017, without
 93269 the `_CS_` prefix.

93270 The names of the symbolic constants listed under the headings “Maximum
93271 Values” and “Minimum Values” in the description of the `<limits.h>` header
93272 in the Base Definitions volume of POSIX.1-2017, without the enclosing
93273 braces.

93274 For compatibility with earlier versions, the following variable names shall
93275 also be supported:

93276 POSIX2_BC_BASE_MAX
93277 POSIX2_BC_DIM_MAX
93278 POSIX2_BC_SCALE_MAX
93279 POSIX2_BC_STRING_MAX
93280 POSIX2_COLL_WEIGHTS_MAX
93281 POSIX2_EXPR_NEST_MAX
93282 POSIX2_LINE_MAX
93283 POSIX2_RE_DUP_MAX

93284 and shall be equivalent to the same name prefixed with an `<underscore>`.
93285 This requirement may be removed in a future version.

93286 The implementation may add other local values.

93287 **STDIN**

93288 Not used.

93289 **INPUT FILES**

93290 None.

93291 **ENVIRONMENT VARIABLES**

93292 The following environment variables shall affect the execution of *getconf*:

93293 *LANG* Provide a default value for the internationalization variables that are unset or null.
93294 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
93295 variables used to determine the values of locale categories.)

93296 *LC_ALL* If set to a non-empty string value, override the values of all the other
93297 internationalization variables.

93298 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
93299 characters (for example, single-byte as opposed to multi-byte characters in
93300 arguments).

93301 *LC_MESSAGES*

93302 Determine the locale that should be used to affect the format and contents of
93303 diagnostic messages written to standard error.

93304 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93305 **ASYNCHRONOUS EVENTS**

93306 Default.

93307 **STDOUT**

93308 If the specified variable is defined on the system and its value is described to be available from
93309 the *confstr()* function defined in the System Interfaces volume of POSIX.1-2017, its value shall be
93310 written in the following format:

93311 "%s\n", *<value>*

93312 Otherwise, if the specified variable is defined on the system, its value shall be written in the

93313 following format:

93314 "%d\n", <value>

93315 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the

93316 following format:

93317 "undefined\n"

93318 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

93319 **STDERR**

93320 The standard error shall be used only for diagnostic messages.

93321 **OUTPUT FILES**

93322 None.

93323 **EXTENDED DESCRIPTION**

93324 None.

93325 **EXIT STATUS**

93326 The following exit values shall be returned:

93327 0 The specified variable is valid and information about its current state was written

93328 successfully.

93329 >0 An error occurred.

93330 **CONSEQUENCES OF ERRORS**

93331 Default.

93332 **APPLICATION USAGE**

93333 None.

93334 **EXAMPLES**

93335 The following example illustrates the value of {NGROUPS_MAX}:

93336 `getconf NGROUPS_MAX`

93337 The following example illustrates the value of {NAME_MAX} for a specific directory:

93338 `getconf NAME_MAX /usr`

93339 The following example shows how to deal more carefully with results that might be unspecified:

93340 `if value=$(getconf PATH_MAX /usr); then`

93341 `if ["$value" = "undefined"]; then`

93342 `echo PATH_MAX in /usr is indeterminate.`

93343 `else`

93344 `echo PATH_MAX in /usr is $value.`

93345 `fi`

93346 `else`

93347 `echo Error in getconf.`

93348 `fi`

93349 **RATIONALE**

93350 The original need for this utility, and for the *confstr()* function, was to provide a way of finding

93351 the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be

93352 modified by the user to include directories that could contain utilities replacing the standard

93353 utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable

93354 value that contains the correct search path for the standard utilities. It was later suggested that

93355 access to the other variables described in this volume of POSIX.1-2017 could also be useful to
93356 applications.

93357 This functionality of *getconf* would not be adequately subsumed by another command such as:

93358 `grep var /etc/conf`

93359 because such a strategy would provide correct values for neither those variables that can vary at
93360 runtime, nor those that can vary depending on the path.

93361 Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid,
93362 but not defined on the system. The output string "undefined" is now used to specify this case
93363 with exit code 0 because so many things depend on an exit code of zero when an invoked utility
93364 is successful.

93365 **FUTURE DIRECTIONS**

93366 None.

93367 **SEE ALSO**

93368 [c99](#)

93369 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216), [<limits.h>](#)

93370 XSH [confstr\(\)](#), [fpathconf\(\)](#), [sysconf\(\)](#), [system\(\)](#)

93371 **CHANGE HISTORY**

93372 First released in Issue 4.

93373 **Issue 5**

93374 In the OPERANDS section:

93375 {NL_MAX} is changed to {NL_NMAX}.

93376 Entries beginning NL_ are deleted from the list of standard configuration variables.

93377 The list of variables previously marked UX is merged with the list marked EX.

93378 Operands are added to support new Option Groups.

93379 Operands are added so that *getconf* can determine supported programming environments.

93380 **Issue 6**

93381 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last
93382 paragraph of the OPTIONS section.

93383 The following new requirements on POSIX implementations derive from alignment with the
93384 Single UNIX Specification:

93385 Operands are added to determine supported programming environments.

93386 This reference page is updated for alignment with the ISO/IEC 9899: 1999 standard. Specifically,
93387 new macros for *c99* programming environments are introduced.

93388 XSI marked *system_var* (XBS5_*) values are marked LEGACY.

93389 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions
93390 of *path_var* and *system_var* in the OPERANDS section.

93391 **Issue 7**

93392 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93393 The EXAMPLES section is corrected.

93394

POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0091 [125] is applied.

93395 **NAME**

93396 getopts ‡parse utility options

93397 **SYNOPSIS**93398 getopts *optstring name* [*arg...*]93399 **DESCRIPTION**93400 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall support the Utility Syntax Guidelines 3 to 10, inclusive, described in XBD [Section 12.2](#) (on page 216).93403 Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell variable specified by the *name* operand and the index of the next argument to be processed in the shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.93406 When the option requires an option-argument, the *getopts* utility shall place it in the shell variable *OPTARG*. If no option was found, or if the option that was found does not have an option-argument, *OPTARG* shall be unset.93409 If an option character not contained in the *optstring* operand is found where an option character is expected, the shell variable specified by *name* shall be set to the <question-mark> ('?') character. In this case, if the first character in *optstring* is a <colon> (':'), the shell variable *OPTARG* shall be set to the option character found, but no output shall be written to standard error; otherwise, the shell variable *OPTARG* shall be unset and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in *getopts* processing.

93417 If an option-argument is missing:

93418 If the first character of *optstring* is a <colon>, the shell variable specified by *name* shall be set to the <colon> character and the shell variable *OPTARG* shall be set to the option character found.93421 Otherwise, the shell variable specified by *name* shall be set to the <question-mark> character, the shell variable *OPTARG* shall be unset, and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in *getopts* processing; a diagnostic message shall be written as stated, but the exit status shall be zero.93427 When the end of options is encountered, the *getopts* utility shall exit with a return value greater than zero; the shell variable *OPTIND* shall be set to the index of the first operand, or the value "\$#+1" if there are no operands; the *name* variable shall be set to the <question-mark> character. Any of the following shall identify the end of options: the first "--" argument that is not an option-argument, finding an argument that is not an option-argument and does not begin with a '-', or encountering an error.93433 The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be exported by default.93435 The shell variable specified by the *name* operand, *OPTIND*, and *OPTARG* shall affect the current shell execution environment; see [Section 2.12](#) (on page 2381).93437 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple times in a single shell execution environment with parameters (positional parameters or *arg* operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a

93441 value other than 1, produces unspecified results.

93442 **OPTIONS**

93443 None.

93444 **OPERANDS**

93445 The following operands shall be supported:

93446 *optstring* A string containing the option characters recognized by the utility invoking *getopts*.
 93447 If a character is followed by a <colon>, the option shall be expected to have an
 93448 argument, which should be supplied as a separate argument. Applications should
 93449 specify an option character and its option-argument as separate arguments, but
 93450 *getopts* shall interpret the characters following an option character requiring
 93451 arguments as an argument whether or not this is done. An explicit null option-
 93452 argument need not be recognized if it is not supplied as a separate argument when
 93453 *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces
 93454 volume of POSIX.1-2017.) The characters <question-mark> and <colon> shall not
 93455 be used as option characters by an application. The use of other option characters
 93456 that are not alphanumeric produces unspecified results. If the option-argument is
 93457 not supplied as a separate argument from the option character, the value in
 93458 *OPTARG* shall be stripped of the option character and the '-'. The first character
 93459 in *optstring* determines how *getopts* behaves if an option character is not known or
 93460 an option-argument is missing.

93461 *name* The name of a shell variable that shall be set by the *getopts* utility to the option
 93462 character that was found.

93463 The *getopts* utility by default shall parse positional parameters passed to the invoking shell
 93464 procedure. If *args* are given, they shall be parsed instead of the positional parameters.

93465 **STDIN**

93466 Not used.

93467 **INPUT FILES**

93468 None.

93469 **ENVIRONMENT VARIABLES**

93470 The following environment variables shall affect the execution of *getopts*:

93471 *LANG* Provide a default value for the internationalization variables that are unset or null.
 93472 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 93473 variables used to determine the values of locale categories.)

93474 *LC_ALL* If set to a non-empty string value, override the values of all the other
 93475 internationalization variables.

93476 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93477 characters (for example, single-byte as opposed to multi-byte characters in
 93478 arguments and input files).

93479 *LC_MESSAGES*

93480 Determine the locale that should be used to affect the format and contents of
 93481 diagnostic messages written to standard error.

93482 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93483 *OPTIND* This variable shall be used by the *getopts* utility as the index of the next argument
 93484 to be processed.

93485 **ASYNCHRONOUS EVENTS**

93486 Default.

93487 **STDOUT**

93488 Not used.

93489 **STDERR**

93490 Whenever an error is detected and the first character in the *optstring* operand is not a <colon>
 93491 (' : '), a diagnostic message shall be written to standard error with the following information in
 93492 an unspecified format:

93493 The invoking program name shall be identified in the message. The invoking program
 93494 name shall be the value of the shell special parameter 0 (see [Section 2.5.2](#), on page 2350) at
 93495 the time the *getopts* utility is invoked. A name equivalent to:

93496 `basename "$0"`

93497 may be used.

93498 If an option is found that was not specified in *optstring*, this error is identified and the
 93499 invalid option character shall be identified in the message.

93500 If an option requiring an option-argument is found, but an option-argument is not found,
 93501 this error shall be identified and the invalid option character shall be identified in the
 93502 message.

93503 **OUTPUT FILES**

93504 None.

93505 **EXTENDED DESCRIPTION**

93506 None.

93507 **EXIT STATUS**

93508 The following exit values shall be returned:

93509 `0` An option, specified or unspecified by *optstring*, was found.93510 `>0` The end of options was encountered or an error occurred.93511 **CONSEQUENCES OF ERRORS**

93512 Default.

93513 **APPLICATION USAGE**

93514 Since *getopts* affects the current shell execution environment, it is generally provided as a shell
 93515 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
 93516 of the following:

93517 `(getopts abc value "$@")`93518 `nohup getopts ...`93519 `find . -exec getopts ... \;`

93520 it does not affect the shell variables in the caller's environment.

93521 Note that shell functions share *OPTIND* with the calling shell even though the positional
 93522 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse
 93523 arguments, the results are unspecified.

93524 **EXAMPLES**

93525 The following example script parses and displays its arguments:

```

93526 aflag=
93527 bflag=
93528 while getopts ab: name
93529 do
93530     case $name in
93531         a)  aflag=1;;
93532         b)  bflag=1
93533             bval="$OPTARG";;
93534         ?)  printf "Usage: %s: [-a] [-b value] args\n" $0
93535             exit 2;;
93536     esac
93537 done
93538 if [ ! -z "$aflag" ]; then
93539     printf "Option -a specified\n"
93540 fi
93541 if [ ! -z "$bflag" ]; then
93542     printf 'Option -b "%s" specified\n' "$bval"
93543 fi
93544 shift $(( $OPTIND - 1 ))
93545 printf "Remaining arguments are: %s\n" "$*"

```

93546 **RATIONALE**

93547 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles
 93548 option-arguments containing <blank> characters.

93549 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it
 93550 does not affect the execution of *getopts*; it is one of the few “output-only” variables used by the
 93551 standard utilities.

93552 The <colon> is not allowed as an option character because that is not historical behavior, and it
 93553 violates the Utility Syntax Guidelines. The <colon> is now specified to behave as in the
 93554 KornShell version of the *getopts* utility; when used as the first character in the *optstring* operand,
 93555 it disables diagnostics concerning missing option-arguments and unexpected option characters.
 93556 This replaces the use of the *OPTERR* variable that was specified in an early proposal.

93557 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function
 93558 are not fully specified because implementations with superior (“friendlier”) formats objected to
 93559 the formats used by some historical implementations. The standard developers considered it
 93560 important that the information in the messages used be uniform between *getopts* and *getopt()*.
 93561 Exact duplication of the messages might not be possible, particularly if a utility is built on
 93562 another system that has a different *getopt()* function, but the messages must have specific
 93563 information included so that the program name, invalid option character, and type of error can
 93564 be distinguished by a user.

93565 Only a rare application program intercepts a *getopts* standard error message and wants to parse
 93566 it. Therefore, implementations are free to choose the most usable messages they can devise. The
 93567 following formats are used by many historical implementations:

```

93568 "%s: illegal option -- %c\n", <program name>, <option character>
93569 "%s: option requires an argument -- %c\n", <program name>, \
93570     <option character>

```

93571 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,
93572 frequently not even indicating the option character found in error.

93573 **FUTURE DIRECTIONS**

93574 None.

93575 **SEE ALSO**

93576 [Section 2.5.2](#) (on page 2350)

93577 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

93578 XSH [getopt\(\)](#)

93579 **CHANGE HISTORY**

93580 First released in Issue 4.

93581 **Issue 6**

93582 The normative text is reworded to avoid use of the term “must” for application requirements.

93583 **Issue 7**

93584 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0092 [159] is applied.

93585 **NAME**93586 `grep` — search a file for a pattern93587 **SYNOPSIS**

```
93588  grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list
93589      [-e pattern_list]... [-f pattern_file]... [file...]
93590  grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...
93591      -f pattern_file [-f pattern_file]... [file...]
93592  grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]
```

93593 **DESCRIPTION**

93594 The `grep` utility shall search the input files, selecting lines matching one or more patterns; the
 93595 types of patterns are controlled by the options specified. The patterns are specified by the `-e`
 93596 option, `-f` option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more
 93597 patterns separated by <newline> characters; the *pattern_file*'s contents shall consist of one or
 93598 more patterns terminated by a <newline> character. By default, an input line shall be selected if
 93599 any pattern, treated as an entire basic regular expression (BRE) as described in XBD [Section 9.3](#)
 93600 (on page 183), matches any part of the line excluding the terminating <newline>; a null BRE
 93601 shall match every line. By default, each selected input line shall be written to the standard
 93602 output.

93603 Regular expression matching shall be based on text lines. Since a <newline> separates or
 93604 terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a
 93605 <newline>. Similarly, since patterns are matched against individual lines (excluding the
 93606 terminating <newline> characters) of the input, there is no way for a pattern to match a
 93607 <newline> found in the input.

93608 **OPTIONS**93609 The `grep` utility shall conform to XBD [Section 12.2](#) (on page 216).

93610 The following options shall be supported:

93611 `-E` Match using extended regular expressions. Treat each pattern specified as an ERE,
 93612 as described in XBD [Section 9.4](#) (on page 188). If any entire ERE pattern matches
 93613 some part of an input line excluding the terminating <newline>, the line shall be
 93614 matched. A null ERE shall match every line.

93615 `-F` Match using fixed strings. Treat each pattern specified as a string instead of a
 93616 regular expression. If an input line contains any of the patterns as a contiguous
 93617 sequence of bytes, the line shall be matched. A null string shall match every line.

93618 `-c` Write only a count of selected lines to standard output.

93619 `-e pattern_list`

93620 Specify one or more patterns to be used during the search for input. The
 93621 application shall ensure that patterns in *pattern_list* are separated by a <newline>.
 93622 A null pattern can be specified by two adjacent <newline> characters in
 93623 *pattern_list*. Unless the `-E` or `-F` option is also specified, each pattern shall be
 93624 treated as a BRE, as described in XBD [Section 9.3](#) (on page 183). Multiple `-e` and `-f`
 93625 options shall be accepted by the `grep` utility. All of the specified patterns shall be
 93626 used when matching lines, but the order of evaluation is unspecified.

93627 `-f pattern_file`

93628 Read one or more patterns from the file named by the pathname *pattern_file*.
 93629 Patterns in *pattern_file* shall be terminated by a <newline>. A null pattern can be
 93630 specified by an empty line in *pattern_file*. Unless the `-E` or `-F` option is also

- 93631 specified, each pattern shall be treated as a BRE, as described in XBD [Section 9.3](#)
93632 (on page 183).
- 93633 **-i** Perform pattern matching in searches without regard to case; see XBD [Section 9.2](#)
93634 (on page 182).
- 93635 **-l** (The letter ell.) Write only the names of files containing selected lines to standard
93636 output. Pathnames shall be written once per file searched. If the standard input is
93637 searched, a pathname of "(standard input)" shall be written, in the POSIX
93638 locale. In other locales, "standard input" may be replaced by something more
93639 appropriate in those locales.
- 93640 **-n** Precede each output line by its relative line number in the file, each file starting at
93641 line 1. The line number counter shall be reset for each file processed.
- 93642 **-q** Quiet. Nothing shall be written to the standard output, regardless of matching
93643 lines. Exit with zero status if an input line is selected.
- 93644 **-s** Suppress the error messages ordinarily written for nonexistent or unreadable files.
93645 Other error messages shall not be suppressed.
- 93646 **-v** Select lines not matching any of the specified patterns. If the **-v** option is not
93647 specified, selected lines shall be those that match any of the specified patterns.
- 93648 **-x** Consider only input lines that use all characters in the line excluding the
93649 terminating <newline> to match an entire fixed string or regular expression to be
93650 matching lines.

93651 OPERANDS

93652 The following operands shall be supported:

- 93653 *pattern_list* Specify one or more patterns to be used during the search for input. This operand
93654 shall be treated as if it were specified as **-e pattern_list**.
- 93655 *file* A pathname of a file to be searched for the patterns. If no *file* operands are
93656 specified, the standard input shall be used.

93657 STDIN

93658 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
93659 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
93660 the standard input shall not be used. See the INPUT FILES section.

93661 INPUT FILES

93662 The input files shall be text files.

93663 ENVIRONMENT VARIABLES

93664 The following environment variables shall affect the execution of *grep*:

- 93665 *LANG* Provide a default value for the internationalization variables that are unset or null.
93666 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
93667 variables used to determine the values of locale categories.)
- 93668 *LC_ALL* If set to a non-empty string value, override the values of all the other
93669 internationalization variables.
- 93670 *LC_COLLATE*
93671 Determine the locale for the behavior of ranges, equivalence classes, and multi-
93672 character collating elements within regular expressions.

93673 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93674 characters (for example, single-byte as opposed to multi-byte characters in
 93675 arguments and input files) and the behavior of character classes within regular
 93676 expressions.

93677 *LC_MESSAGES*
 93678 Determine the locale that should be used to affect the format and contents of
 93679 diagnostic messages written to standard error.

93680 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93681 ASYNCHRONOUS EVENTS

93682 Default.

93683 STDOUT

93684 If the **-l** option is in effect, the following shall be written for each file containing at least one
 93685 selected input line:

93686 "%s\n", <file>

93687 Otherwise, if more than one *file* argument appears, and **-q** is not specified, the *grep* utility shall
 93688 prefix each output line by:

93689 "%s: ", <file>

93690 The remainder of each output line shall depend on the other options specified:

93691 If the **-c** option is in effect, the remainder of each output line shall contain:

93692 "%d\n", <count>

93693 Otherwise, if **-c** is not in effect and the **-n** option is in effect, the following shall be written
 93694 to standard output:

93695 "%d:", <line number>

93696 Finally, the following shall be written to standard output:

93697 "%s", <selected-line contents>

93698 STDERR

93699 The standard error shall be used only for diagnostic messages.

93700 OUTPUT FILES

93701 None.

93702 EXTENDED DESCRIPTION

93703 None.

93704 EXIT STATUS

93705 The following exit values shall be returned:

93706 0 One or more lines were selected.

93707 1 No lines were selected.

93708 >1 An error occurred.

93709 CONSEQUENCES OF ERRORS

93710 If the **-q** option is specified, the exit status shall be zero if an input line is selected, even if an
 93711 error was detected. Otherwise, default actions shall be performed.

93712 **APPLICATION USAGE**

93713 Care should be taken when using characters in *pattern_list* that may also be meaningful to the
 93714 command interpreter. It is safest to enclose the entire *pattern_list* argument in single-quotes:

93715 ' . . . '

93716 The `-e pattern_list` option has the same effect as the *pattern_list* operand, but is useful when
 93717 *pattern_list* begins with the <hyphen-minus> delimiter. It is also useful when it is more
 93718 convenient to provide multiple patterns as separate arguments.

93719 Multiple `-e` and `-f` options are accepted and *grep* uses all of the patterns it is given while
 93720 matching input text lines. (Note that the order of evaluation is not specified. If an
 93721 implementation finds a null string as a pattern, it is allowed to use that pattern first, matching
 93722 every line, and effectively ignore any other patterns.)

93723 The `-q` option provides a means of easily determining whether or not a pattern (or string) exists
 93724 in a group of files. When searching several files, it provides a performance improvement
 93725 (because it can quit as soon as it finds the first match) and requires less care by the user in
 93726 choosing the set of files to supply as arguments (because it exits zero if it finds a match even if
 93727 *grep* detected an access or read error on earlier *file* operands).

93728 When using *grep* to process pathnames, it is recommended that `LC_ALL`, or at least `LC_CTYPE`
 93729 and `LC_COLLATE`, are set to `POSIX` or `C` in the environment, since pathnames can contain byte
 93730 sequences that do not form valid characters in some locales, in which case the utility's behavior
 93731 would be undefined. In the `POSIX` locale each byte is a valid single-byte character, and therefore
 93732 this problem is avoided.

93733 **EXAMPLES**

93734 1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line
 93735 numbers:

93736 `grep -i -n posix text.mm`

93737 2. To find all empty lines in the standard input:

93738 `grep ^$`

93739 or:

93740 `grep -v .`

93741 3. Both of the following commands print all lines containing strings "abc" or "def" or
 93742 both:

93743 `grep -E 'abc|def'`

93744 `grep -F 'abc`

93745 `def'`

93746 4. Both of the following commands print all lines matching exactly "abc" or "def":

93747 `grep -E '^abc$|^def$'`

93748 `grep -F -x 'abc`

93749 `def'`

93750 **RATIONALE**

93751 This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of
93752 the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard
93753 developers to consolidate the three *greps* into a single command.

93754 The old *egrep* and *fgrep* commands are likely to be supported for many years to come as
93755 implementation extensions, allowing historical applications to operate unmodified.

93756 Historical implementations usually silently ignored all but one of multiply-specified *-e* and *-f*
93757 options, but were not consistent as to which specification was actually used.

93758 The *-b* option was omitted from the OPTIONS section because block numbers are
93759 implementation-defined.

93760 The System V restriction on using *-* to mean standard input was omitted.

93761 A definition of action taken when given a null BRE or ERE is specified. This is an error
93762 condition in some historical implementations.

93763 The *-I* option previously indicated that its use was undefined when no files were explicitly
93764 named. This behavior was historical and placed an unnecessary restriction on future
93765 implementations. It has been removed.

93766 The historical BSD *grep -s* option practice is easily duplicated by redirecting standard output to
93767 */dev/null*. The *-s* option required here is from System V.

93768 The *-x* option, historically available only with *fgrep*, is available here for all of the non-
93769 obsolescent versions.

93770 **FUTURE DIRECTIONS**

93771 None.

93772 **SEE ALSO**

93773 *sed*

93774 XBD [Chapter 8](#) (on page 173), [Chapter 9](#) (on page 181), [Section 12.2](#) (on page 216)

93775 **CHANGE HISTORY**

93776 First released in Issue 2.

93777 **Issue 6**

93778 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.

93779 The normative text is reworded to avoid use of the term “must” for application requirements.

93780 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples
93781 using the *grep -F* option which did not match the normative description of the *-F* option.

93782 **Issue 7**

93783 Austin Group Interpretation 1003.1-2001 #092 is applied.

93784 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93785 SD5-XCU-ERN-98 is applied, updating the STDOUT section.

93786 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0105 [584] and XCU/TC2-2008/0106
93787 [663] are applied.

93788 **NAME**

93789 hash — remember or report utility locations

93790 **SYNOPSIS**93791 hash [*utility*...]

93792 hash -r

93793 **DESCRIPTION**

93794 The *hash* utility shall affect the way the current shell environment remembers the locations of
 93795 utilities found as described in [Section 2.9.1.1](#) (on page 2367). Depending on the arguments
 93796 specified, it shall add utility locations to its list of remembered locations or it shall purge the
 93797 contents of the list. When no arguments are specified, it shall report on the contents of the list.

93798 Utilities provided as built-ins to the shell shall not be reported by *hash*.93799 **OPTIONS**93800 The *hash* utility shall conform to XBD [Section 12.2](#) (on page 216).

93801 The following option shall be supported:

93802 -r Forget all previously remembered utility locations.

93803 **OPERANDS**

93804 The following operand shall be supported:

93805 *utility* The name of a utility to be searched for and added to the list of remembered
 93806 locations. If *utility* contains one or more <slash> characters, the results are
 93807 unspecified.

93808 **STDIN**

93809 Not used.

93810 **INPUT FILES**

93811 None.

93812 **ENVIRONMENT VARIABLES**93813 The following environment variables shall affect the execution of *hash*:

93814 *LANG* Provide a default value for the internationalization variables that are unset or null.
 93815 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 93816 variables used to determine the values of locale categories.)

93817 *LC_ALL* If set to a non-empty string value, override the values of all the other
 93818 internationalization variables.

93819 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 93820 characters (for example, single-byte as opposed to multi-byte characters in
 93821 arguments).

93822 *LC_MESSAGES*

93823 Determine the locale that should be used to affect the format and contents of
 93824 diagnostic messages written to standard error.

93825 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.93826 *PATH* Determine the location of *utility*, as described in XBD [Chapter 8](#) (on page 173).

93827 **ASYNCHRONOUS EVENTS**

93828 Default.

93829 **STDOUT**

93830 The standard output of *hash* shall be used when no arguments are specified. Its format is
93831 unspecified, but includes the pathname of each utility in the list of remembered locations for the
93832 current shell environment. This list shall consist of those utilities named in previous *hash*
93833 invocations that have been invoked, and may contain those invoked and found through the
93834 normal command search process.

93835 **STDERR**

93836 The standard error shall be used only for diagnostic messages.

93837 **OUTPUT FILES**

93838 None.

93839 **EXTENDED DESCRIPTION**

93840 None.

93841 **EXIT STATUS**

93842 The following exit values shall be returned:

93843 0 Successful completion.

93844 >0 An error occurred.

93845 **CONSEQUENCES OF ERRORS**

93846 Default.

93847 **APPLICATION USAGE**

93848 Since *hash* affects the current shell execution environment, it is always provided as a shell
93849 regular built-in. If it is called in a separate utility execution environment, such as one of the
93850 following:

```
93851 nohup hash -r  
93852 find . -type f | xargs hash
```

93853 it does not affect the command search process of the caller's environment.

93854 The *hash* utility may be implemented as an alias `⚭for example alias -t -`, in which case utilities
93855 found through normal command search are not listed by the *hash* command.

93856 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the
93857 simplest form, this can be:

93858 `PATH="$PATH"`

93859 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a
93860 performance improvement on a few implementations; normally, the hashing process is included
93861 by default.

93862 **EXAMPLES**

93863 None.

93864 **RATIONALE**

93865 None.

93866 **FUTURE DIRECTIONS**

93867 None.

93868 **SEE ALSO**93869 [Section 2.9.1.1](#) (on page 2367)93870 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)93871 **CHANGE HISTORY**

93872 First released in Issue 2.

93873 **Issue 7**93874 The *hash* utility is moved from the XSI option to the Base.

93875 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0093 [241] is applied.

93876 **NAME**

93877 head ‡copy the first part of files

93878 **SYNOPSIS**93879 head [-n *number*] [*file...*]93880 **DESCRIPTION**93881 The *head* utility shall copy its input files to the standard output, ending the output for each file at
93882 a designated point.93883 Copying shall end at the point in each input file indicated by the *-n number* option. The option-
93884 argument *number* shall be counted in units of lines.93885 **OPTIONS**93886 The *head* utility shall conform to XBD [Section 12.2](#) (on page 216).

93887 The following option shall be supported:

93888 *-n number* The first *number* lines of each input file shall be copied to standard output. The
93889 application shall ensure that the *number* option-argument is a positive decimal
93890 integer.93891 When a file contains less than *number* lines, it shall be copied to standard output in its entirety.
93892 This shall not be an error.93893 If no options are specified, *head* shall act as if *-n 10* had been specified.93894 **OPERANDS**

93895 The following operand shall be supported:

93896 *file* A pathname of an input file. If no *file* operands are specified, the standard input
93897 shall be used.93898 **STDIN**93899 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
93900 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
93901 the standard input shall not be used. See the INPUT FILES section.93902 **INPUT FILES**

93903 Input files shall be text files, but the line length is not restricted to {LINE_MAX} bytes.

93904 **ENVIRONMENT VARIABLES**93905 The following environment variables shall affect the execution of *head*:93906 *LANG* Provide a default value for the internationalization variables that are unset or null.
93907 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
93908 variables used to determine the values of locale categories.)93909 *LC_ALL* If set to a non-empty string value, override the values of all the other
93910 internationalization variables.93911 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
93912 characters (for example, single-byte as opposed to multi-byte characters in
93913 arguments and input files).93914 *LC_MESSAGES*93915 Determine the locale that should be used to affect the format and contents of
93916 diagnostic messages written to standard error.

93917 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

93918 **ASYNCHRONOUS EVENTS**

93919 Default.

93920 **STDOUT**

93921 The standard output shall contain designated portions of the input files.

93922 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

93923 "\n==> %s <==\n", <pathname>

93924 except that the first header written shall not include the initial <newline>.

93925 **STDERR**

93926 The standard error shall be used only for diagnostic messages.

93927 **OUTPUT FILES**

93928 None.

93929 **EXTENDED DESCRIPTION**

93930 None.

93931 **EXIT STATUS**

93932 The following exit values shall be returned:

93933 0 Successful completion.

93934 >0 An error occurred.

93935 **CONSEQUENCES OF ERRORS**

93936 Default.

93937 **APPLICATION USAGE**

93938 When using *head* to process pathnames, it is recommended that *LC_ALL*, or at least *LC_CTYPE*
 93939 and *LC_COLLATE*, are set to *POSIX* or *C* in the environment, since pathnames can contain byte
 93940 sequences that do not form valid characters in some locales, in which case the utility's behavior
 93941 would be undefined. In the *POSIX* locale each byte is a valid single-byte character, and therefore
 93942 this problem is avoided.

93943 **EXAMPLES**

93944 To write the first ten lines of all files (except those with a leading period) in the directory:

93945 `head -- *`

93946 **RATIONALE**

93947 Although it is possible to simulate *head* with *sed 10q* for a single file, the standard developers
 93948 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside
 93949 *tail*.

93950 *POSIX.1-2017* version of *head* follows the Utility Syntax Guidelines. The *-n* option was added to
 93951 this new interface so that *head* and *tail* would be more logically related. Earlier versions of this
 93952 standard allowed a *-number* option. This form is no longer specified by *POSIX.1-2017* but may
 93953 be present in some implementations.

93954 There is no *-c* option (as there is in *tail*) because it is not historical practice and because other
 93955 utilities in this volume of *POSIX.1-2017* provide similar functionality.

93956 **FUTURE DIRECTIONS**

93957 None.

93958 **SEE ALSO**93959 *sed, tail*93960 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)93961 **CHANGE HISTORY**

93962 First released in Issue 4.

93963 **Issue 6**93964 The obsolescent **-number** form is removed.

93965 The normative text is reworded to avoid use of the term “must” for application requirements.

93966 The DESCRIPTION is updated to clarify that when a file contains less than the number of lines requested, the entire file is copied to standard output.
9396793968 **Issue 7**

93969 Austin Group Interpretations 1003.1-2001 #027 and #092 are applied.

93970 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93971 The APPLICATION USAGE section is removed and the EXAMPLES section is corrected.

93972 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0107 [663] is applied.

93973 **NAME**93974 iconv \ddagger 'codeset conversion93975 **SYNOPSIS**93976 iconv [-cs] -f *frommap* -t *tomap* [*file...*]93977 iconv -f *fromcode* [-cs] [-t *tocode*] [*file...*]93978 iconv -t *tocode* [-cs] [-f *fromcode*] [*file...*]

93979 iconv -l

93980 **DESCRIPTION**93981 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and
93982 write the results to standard output.93983 When the options indicate that charmap files are used to specify the codesets (see **OPTIONS**),
93984 the codeset conversion shall be accomplished by performing a logical join on the symbolic
93985 character names in the two charmaps. The implementation need not support the use of charmap
93986 files for codeset conversion unless the POSIX2_LOCALEDEF symbol is defined on the system.93987 **OPTIONS**93988 The *iconv* utility shall conform to XBD [Section 12.2](#) (on page 216).

93989 The following options shall be supported:

93990 **-c** Omit any characters that are invalid in the codeset of the input file from the
93991 output. When **-c** is not used, the results of encountering invalid characters in the
93992 input stream (either those that are not characters in the codeset of the input file or
93993 that have no corresponding character in the codeset of the output file) shall be
93994 specified in the system documentation. The presence or absence of **-c** shall not
93995 affect the exit status of *iconv*.93996 **-f** *fromcodeset*93997 Identify the codeset of the input file. The implementation shall recognize the
93998 following two forms of the *fromcodeset* option-argument:93999 *fromcode* The *fromcode* option-argument must not contain a <slash> character.
94000 It shall be interpreted as the name of one of the codeset descriptions
94001 provided by the implementation in an unspecified format. Valid
94002 values of *fromcode* are implementation-defined.94003 *frommap* The *frommap* option-argument must contain a <slash> character. It
94004 shall be interpreted as the pathname of a charmap file as defined in
94005 XBD [Section 6.4](#) (on page 129). If the pathname does not represent a
94006 valid, readable charmap file, the results are undefined.

94007 If this option is omitted, the codeset of the current locale shall be used.

94008 **-l** Write all supported *fromcode* and *tocode* values to standard output in an unspecified
94009 format.94010 **-s** Suppress any messages written to standard error concerning invalid characters.
94011 When **-s** is not used, the results of encountering invalid characters in the input
94012 stream (either those that are not valid characters in the codeset of the input file or
94013 that have no corresponding character in the codeset of the output file) shall be
94014 specified in the system documentation. The presence or absence of **-s** shall not
94015 affect the exit status of *iconv*.

94016 **-t tocodeset** Identify the codeset to be used for the output file. The implementation shall
94017 recognize the following two forms of the *tocodeset* option-argument:

94018 *toctype* The semantics shall be equivalent to the *-f fromcode* option.

94019 *tomap* The semantics shall be equivalent to the *-f frommap* option.

94020 If this option is omitted, the codeset of the current locale shall be used.

94021 If either *-f* or *-t* represents a charmap file, but the other does not (or is omitted), or both *-f* and
94022 *-t* are omitted, the results are undefined.

94023 **OPERANDS**

94024 The following operand shall be supported:

94025 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
94026 '*-*', the standard input shall be used.

94027 **STDIN**

94028 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '*-*'.

94029 **INPUT FILES**

94030 The input file shall be a text file.

94031 **ENVIRONMENT VARIABLES**

94032 The following environment variables shall affect the execution of *iconv*:

94033 *LANG* Provide a default value for the internationalization variables that are unset or null.
94034 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
94035 variables used to determine the values of locale categories.)

94036 *LC_ALL* If set to a non-empty string value, override the values of all the other
94037 internationalization variables.

94038 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
94039 characters (for example, single-byte as opposed to multi-byte characters in
94040 arguments). During translation of the file, this variable is superseded by the use of
94041 the *fromcode* option-argument.

94042 *LC_MESSAGES*

94043 Determine the locale that should be used to affect the format and contents of
94044 diagnostic messages written to standard error.

94045 **XSI** *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

94046 **ASYNCHRONOUS EVENTS**

94047 Default.

94048 **STDOUT**

94049 When the *-l* option is used, the standard output shall contain all supported *fromcode* and *toctype*
94050 values, written in an unspecified format.

94051 When the *-l* option is not used, the standard output shall contain the sequence of characters
94052 read from the input files, translated to the specified codeset. Nothing else shall be written to the
94053 standard output.

94054 **STDERR**

94055 The standard error shall be used only for diagnostic messages.

94056 OUTPUT FILES

94057 None.

94058 EXTENDED DESCRIPTION

94059 None.

94060 EXIT STATUS

94061 The following exit values shall be returned:

94062 0 Successful completion.

94063 >0 An error occurred.

94064 CONSEQUENCES OF ERRORS

94065 Default.

94066 APPLICATION USAGE

94067 The user must ensure that both charmap files use the same symbolic names for characters the
94068 two codesets have in common.

94069 EXAMPLES

94070 The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:2001
94071 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file
94072 **mail.local**:

94073 `iconv -f IS6937 -t IS8859 mail.x400 > mail.local`

94074 RATIONALE

94075 The *iconv* utility can be used portably only when the user provides two charmap files as option-
94076 arguments. This is because a single charmap provided by the user cannot reliably be joined with
94077 the names in a system-provided character set description. The valid values for *fromcode* and
94078 *toctype* are implementation-defined and do not have to have any relation to the charmap
94079 mechanisms. As an aid to interactive users, the `-I` option was adopted from the Plan 9 operating
94080 system. It writes information concerning these implementation-defined values. The format is
94081 unspecified because there are many possible useful formats that could be chosen, such as a
94082 matrix of valid combinations of *fromcode* and *toctype*. The `-I` option is not intended for shell script
94083 usage; conforming applications will have to use charmaps.

94084 The *iconv* utility may support the conversion between ASCII and EBCDIC-based encodings, but
94085 is not required to do so. In an XSI-compliant implementation, the *dd* utility is the only method
94086 guaranteed to support conversion between these two character sets.

94087 FUTURE DIRECTIONS

94088 None.

94089 SEE ALSO

94090 *dd*, *gencat*

94091 XBD [Section 6.4](#) (on page 129), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

94092 CHANGE HISTORY

94093 First released in Issue 3.

94094 Issue 6

94095 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the
94096 ability to use charmap files for conversion has been added.

94097 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address
94098 inconsistencies with the *iconv()* function in the System Interfaces volume of POSIX.1-2017.

94099 **Issue 7**

94100 Austin Group Interpretation 1003.1-2001 #206 is applied, correcting the *tomap* option.

94101 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94102 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0094 [291] and XCU/TC1-2008/0095
94103 [291] are applied.

94104 **NAME**94105 `id` — return user identity94106 **SYNOPSIS**94107 `id` [*user*]94108 `id -G` [-n] [*user*]94109 `id -g` [-nr] [*user*]94110 `id -u` [-nr] [*user*]94111 **DESCRIPTION**

94112 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the
 94113 corresponding user and group names of the invoking process to standard output. If the effective
 94114 and real IDs do not match, both shall be written. If multiple groups are supported by the
 94115 underlying system (see the description of {NGROUPS_MAX} in the System Interfaces volume of
 94116 POSIX.1-2017), the supplementary group affiliations of the invoking process shall also be
 94117 written.

94118 If a *user* operand is provided and the process has appropriate privileges, the user and group IDs
 94119 of the selected user shall be written. In this case, effective IDs shall be assumed to be identical to
 94120 real IDs. If the selected user has more than one allowable group membership listed in the group
 94121 database, these shall be written in the same manner as the supplementary groups described in
 94122 the preceding paragraph.

94123 **OPTIONS**94124 The *id* utility shall conform to XBD [Section 12.2](#) (on page 216).

94125 The following options shall be supported:

94126 **-G** Output all different group IDs (effective, real, and supplementary) only, using the
 94127 format "%u\n". If there is more than one distinct group affiliation, output each
 94128 such affiliation, using the format " %u", before the <newline> is output.

94129 **-g** Output only the effective group ID, using the format "%u\n".

94130 **-n** Output the name in the format "%s" instead of the numeric ID using the format
 94131 "%u".

94132 **-r** Output the real ID instead of the effective ID.

94133 **-u** Output only the effective user ID, using the format "%u\n".

94134 **OPERANDS**

94135 The following operand shall be supported:

94136 *user* The login name for which information is to be written.

94137 **STDIN**

94138 Not used.

94139 **INPUT FILES**

94140 None.

94141 **ENVIRONMENT VARIABLES**94142 The following environment variables shall affect the execution of *id*:

94143 *LANG* Provide a default value for the internationalization variables that are unset or null.
 94144 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 94145 variables used to determine the values of locale categories.)

94146 *LC_ALL* If set to a non-empty string value, override the values of all the other
94147 internationalization variables.

94148 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
94149 characters (for example, single-byte as opposed to multi-byte characters in
94150 arguments).

94151 *LC_MESSAGES*
94152 Determine the locale that should be used to affect the format and contents of
94153 diagnostic messages written to standard error and informative messages written to
94154 standard output.

94155 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

94156 **ASYNCHRONOUS EVENTS**
94157 Default.

94158 **STDOUT**
94159 The following formats shall be used when the *LC_MESSAGES* locale category specifies the
94160 POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with
94161 more appropriate strings corresponding to the locale.

94162 "uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,
94163 <real group ID>, <group-name>

94164 If the effective and real user IDs do not match, the following shall be inserted immediately
94165 before the '\n' character in the previous format:

94166 " euid=%u(%s) "

94167 with the following arguments added at the end of the argument list:

94168 <effective user ID>, <effective user-name>

94169 If the effective and real group IDs do not match, the following shall be inserted directly before
94170 the '\n' character in the format string (and after any addition resulting from the effective and
94171 real user IDs not matching):

94172 " egid=%u(%s) "

94173 with the following arguments added at the end of the argument list:

94174 <effective group-ID>, <effective group name>

94175 If the process has supplementary group affiliations or the selected user is allowed to belong to
94176 multiple groups, the first shall be added directly before the <newline> in the format string:

94177 " groups=%u(%s) "

94178 with the following arguments added at the end of the argument list:

94179 <supplementary group ID>, <supplementary group name>

94180 and the necessary number of the following added after that for any remaining supplementary
94181 group IDs:

94182 ", %u(%s) "

94183 and the necessary number of the following arguments added at the end of the argument list:

94184 <supplementary group ID>, <supplementary group name>

94185 If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple

94186 group IDs cannot be mapped by the system into printable user or group names, the
94187 corresponding "(%s)" and *name* argument shall be omitted from the corresponding format
94188 string.

94189 When any of the options are specified, the output format shall be as described in the OPTIONS
94190 section.

94191 **STDERR**

94192 The standard error shall be used only for diagnostic messages.

94193 **OUTPUT FILES**

94194 None.

94195 **EXTENDED DESCRIPTION**

94196 None.

94197 **EXIT STATUS**

94198 The following exit values shall be returned:

94199 0 Successful completion.

94200 >0 An error occurred.

94201 **CONSEQUENCES OF ERRORS**

94202 Default.

94203 **APPLICATION USAGE**

94204 Output produced by the **-G** option and by the default case could potentially produce very long
94205 lines on systems that support large numbers of supplementary groups. (On systems with user
94206 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per
94207 name, 93 supplementary groups plus distinct effective and real group and user IDs could
94208 theoretically overflow the 2048-byte {LINE_MAX} text file line limit on the default output case.
94209 It would take about 186 supplementary groups to overflow the 2048-byte barrier using *id -G*.
94210 This is not expected to be a problem in practice, but in cases where it is a concern, applications
94211 should consider using *fold -s* before post-processing the output of *id*.

94212 **EXAMPLES**

94213 None.

94214 **RATIONALE**

94215 The functionality provided by the 4 BSD *groups* utility can be simulated using:

```
94216 id -Gn [ user ]
```

94217 The 4 BSD command *groups* was considered, but it was not included because it did not provide
94218 the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to
94219 modify *id* to provide the additional functionality necessary to systems with multiple groups than
94220 to invent another command.

94221 The options **-u**, **-g**, **-n**, and **-r** were added to ease the use of *id* with shell commands
94222 substitution. Without these options it is necessary to use some preprocessor such as *sed* to select
94223 the desired piece of information. Since output such as that produced by:

```
94224 id -u -n
```

94225 is frequently wanted, it seemed desirable to add the options.

94226 **FUTURE DIRECTIONS**

94227 None.

94228 **SEE ALSO**94229 *fold, logname, who*94230 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)94231 XSH *getgid()*, *getgroups()*, *getuid()*94232 **CHANGE HISTORY**

94233 First released in Issue 2.

94234 **Issue 7**

94235 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94236 **NAME**

94237 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

94238 **SYNOPSIS**

94239 XSI ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey]...

94240 **DESCRIPTION**94241 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory
94242 segments. The interprocess communication facilities to be removed are specified by the options.94243 Only a user with appropriate privileges shall be allowed to remove an interprocess
94244 communication facility that was not created by or owned by the user invoking *ipcrm*.94245 **OPTIONS**94246 The *ipcrm* utility shall conform to XBD [Section 12.2](#) (on page 216).

94247 The following options shall be supported:

94248 **-q msgid** Remove the message queue identifier *msgid* from the system and destroy the
94249 message queue and data structure associated with it.94250 **-m shmid** Remove the shared memory identifier *shmid* from the system. The shared memory
94251 segment and data structure associated with it shall be destroyed after the last
94252 detach.94253 **-s semid** Remove the semaphore identifier *semid* from the system and destroy the set of
94254 semaphores and data structure associated with it.94255 **-Q msgkey** Remove the message queue identifier, created with key *msgkey*, from the system
94256 and destroy the message queue and data structure associated with it.94257 **-M shmkey** Remove the shared memory identifier, created with key *shmkey*, from the system.
94258 The shared memory segment and data structure associated with it shall be
94259 destroyed after the last detach.94260 **-S semkey** Remove the semaphore identifier, created with key *semkey*, from the system and
94261 destroy the set of semaphores and data structure associated with it.94262 **OPERANDS**

94263 None.

94264 **STDIN**

94265 Not used.

94266 **INPUT FILES**

94267 None.

94268 **ENVIRONMENT VARIABLES**94269 The following environment variables shall affect the execution of *ipcrm*:94270 **LANG** Provide a default value for the internationalization variables that are unset or null.
94271 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
94272 variables used to determine the values of locale categories.)94273 **LC_ALL** If set to a non-empty string value, override the values of all the other
94274 internationalization variables.94275 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
94276 characters (for example, single-byte as opposed to multi-byte characters in
94277 arguments).

- 94278 *LC_MESSAGES*
- 94279 Determine the locale that should be used to affect the format and contents of
94280 diagnostic messages written to standard error.
- 94281 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 94282 **ASYNCHRONOUS EVENTS**
- 94283 Default.
- 94284 **STDOUT**
- 94285 Not used.
- 94286 **STDERR**
- 94287 The standard error shall be used only for diagnostic messages.
- 94288 **OUTPUT FILES**
- 94289 None.
- 94290 **EXTENDED DESCRIPTION**
- 94291 None.
- 94292 **EXIT STATUS**
- 94293 The following exit values shall be returned:
- 94294 0 Successful completion.
- 94295 >0 An error occurred.
- 94296 **CONSEQUENCES OF ERRORS**
- 94297 Default.
- 94298 **APPLICATION USAGE**
- 94299 None.
- 94300 **EXAMPLES**
- 94301 None.
- 94302 **RATIONALE**
- 94303 None.
- 94304 **FUTURE DIRECTIONS**
- 94305 None.
- 94306 **SEE ALSO**
- 94307 *ipcs*
- 94308 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 94309 XSH *msgctl()*, *semctl()*, *shmctl()*
- 94310 **CHANGE HISTORY**
- 94311 First released in Issue 5.
- 94312 **Issue 7**
- 94313 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94314 **NAME**

94315 ipcs — report XSI interprocess communication facilities status

94316 **SYNOPSIS**94317 XSI `ipcs [-qms] [-a|-bcopt]`94318 **DESCRIPTION**94319 The *ipcs* utility shall write information about active interprocess communication facilities.

94320 Without options, information shall be written in short format for message queues, shared
 94321 memory segments, and semaphore sets that are currently active in the system. Otherwise, the
 94322 information that is displayed is controlled by the options specified.

94323 **OPTIONS**94324 The *ipcs* utility shall conform to XBD [Section 12.2](#) (on page 216).94325 The *ipcs* utility accepts the following options:94326 **-q** Write information about active message queues.94327 **-m** Write information about active shared memory segments.94328 **-s** Write information about active semaphore sets.

94329 If **-q**, **-m**, or **-s** are specified, only information about those facilities shall be written. If none of
 94330 these three are specified, information about all three shall be written subject to the following
 94331 options:

94332 **-a** Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

94333 **-b** Write information on maximum allowable size. (Maximum number of bytes in
 94334 messages on queue for message queues, size of segments for shared memory, and
 94335 number of semaphores in each set for semaphores.)

94336 **-c** Write creator's user name and group name; see below.

94337 **-o** Write information on outstanding usage. (Number of messages on queue and total
 94338 number of bytes in messages on queue for message queues, and number of
 94339 processes attached to shared memory segments.)

94340 **-p** Write process number information. (Process ID of the last process to send a
 94341 message and process ID of the last process to receive a message on message
 94342 queues, process ID of the creating process, and process ID of the last process to
 94343 attach or detach on shared memory segments.)

94344 **-t** Write time information. (Time of the last control operation that changed the access
 94345 permissions for all facilities, time of the last *msgsnd()* and *msgrcv()* operations on
 94346 message queues, time of the last *shmat()* and *shmdt()* operations on shared
 94347 memory, and time of the last *semop()* operation on semaphores.)

94348 **OPERANDS**

94349 None.

94350 **STDIN**

94351 Not used.

94352 **INPUT FILES**

94353 The group database

94354 The user database

94355 ENVIRONMENT VARIABLES

94356 The following environment variables shall affect the execution of *ipcs*:

94357 *LANG* Provide a default value for the internationalization variables that are unset or null.
94358 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
94359 variables used to determine the values of locale categories.)

94360 *LC_ALL* If set to a non-empty string value, override the values of all the other
94361 internationalization variables.

94362 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
94363 characters (for example, single-byte as opposed to multi-byte characters in
94364 arguments).

94365 *LC_MESSAGES*
94366 Determine the locale that should be used to affect the format and contents of
94367 diagnostic messages written to standard error.

94368 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

94369 *TZ* Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset
94370 or null, an unspecified default timezone shall be used.

94371 ASYNCHRONOUS EVENTS

94372 Default.

94373 STDOUT

94374 An introductory line shall be written with the format:

94375 `"IPC status from %s as of %s\n", <source>, <date>`

94376 where *<source>* indicates the source used to gather the statistics and *<date>* is the information
94377 that would be produced by the *date* command when invoked in the POSIX locale.

94378 The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options.
94379 The first report shall indicate the status of message queues, the second report shall indicate the
94380 status of shared memory segments, and the third report shall indicate the status of semaphore
94381 sets.

94382 If the corresponding facility is not installed or has not been used since the last reboot, then the
94383 report shall be written out in the format:

94384 `"%s facility not in system.\n", <facility>`

94385 where *<facility>* is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has
94386 been installed and has been used since the last reboot, column headings separated by one or
94387 more *<space>* characters and followed by a *<newline>* shall be written as indicated below
94388 followed by the facility name written out using the format:

94389 `"%s:\n", <facility>`

94390 where *<facility>* is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second
94391 and third reports the column headings need not be written if the last column headings written
94392 already provide column headings for all information in that report.

94393 The column headings provided in the first column below and the meaning of the information in
94394 those columns shall be given in order below; the letters in parentheses indicate the options that
94395 shall cause the corresponding column to appear; "all" means that the column shall always
94396 appear. Each column is separated by one or more *<space>* characters. Note that these options

94397		only determine what information is provided for each report; they do not determine which reports are written.
94398		
94399	T (all)	Type of facility:
94400		q Message queue.
94401		m Shared memory segment.
94402		s Semaphore.
94403		This field is a single character written using the format %c.
94404	ID (all)	The identifier for the facility entry. This field shall be written using the format %d.
94405		
94406	KEY (all)	The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the facility entry.
94407		
94408		Note: The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.
94409		
94410		
94411		This field shall be written using the format 0x%x.
94412	MODE (all)	The facility access modes and flags. The mode shall consist of 11 characters that are interpreted as follows.
94413		
94414		The first character shall be:
94415		S If a process is waiting on a <i>msgsnd()</i> operation.
94416		– If the above is not true.
94417		The second character shall be:
94418		R If a process is waiting on a <i>msgrcv()</i> operation.
94419		C or – If the associated shared memory segment is to be cleared when the first attach operation is executed.
94420		
94421		– If none of the above is true.
94422		The next nine characters shall be interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the usergroup of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is a <hyphen-minus> ('-').
94423		
94424		
94425		
94426		
94427		
94428		The permissions shall be indicated as follows:
94429		r If read permission is granted.
94430		w If write permission is granted.
94431		a If alter permission is granted.
94432		– If the indicated permission is not granted.
94433		The first character following the permissions specifies if there is an alternate or additional access control method associated with the facility. If there is no alternate or additional access control method associated with the facility, a single <space> shall be written; otherwise, another printable character is
94434		
94435		
94436		

94437		written.
94438	OWNER (all)	The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the owner shall be written using the format %d.
94439		
94440		
94441		
94442	GROUP (all)	The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the owner shall be written using the format %d.
94443		
94444		
94445		
94446		The following nine columns shall be only written out for message queues:
94447	CREATOR (a,c)	The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.
94448		
94449		
94450		
94451	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.
94452		
94453		
94454		
94455	CBYTES (a,o)	The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format %d.
94456		
94457	QNUM (a,o)	The number of messages currently outstanding on the associated message queue. This field shall be written using the format %d.
94458		
94459	QBYTES (a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format %d.
94460		
94461	LSPID (a,p)	The process ID of the last process to send a message to the associated queue. This field shall be written using the format:
94462		
94463		"%d", <pid>
94464		where <pid> is 0 if no message has been sent to the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to send a message to the queue.
94465		
94466		
94467	LRPID (a,p)	The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:
94468		
94469		"%d", <pid>
94470		where <pid> is 0 if no message has been received from the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to receive a message from the queue.
94471		
94472		
94473	STIME (a,t)	The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.
94474		
94475		
94476		
94477		
94478	RTIME (a,t)	The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue
94479		
94480		

94481 shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format
94482 " no-entry" shall be written.

94483 The following eight columns shall be only written out for shared memory segments.

94484 CREATOR (a,c) The user of the creator of the facility entry. If the user name of the creator is
94485 found in the user database, at least the first eight column positions of the
94486 name shall be written using the format %s. Otherwise, the user ID of the
94487 creator shall be written using the format %d.

94488 CGROUP (a,c) The group name of the creator of the facility entry. If the group name of the
94489 creator is found in the group database, at least the first eight column positions
94490 of the name shall be written using the format %s. Otherwise, the group ID of
94491 the creator shall be written using the format %d.

94492 NATTCH (a,o) The number of processes attached to the associated shared memory segment.
94493 This field shall be written using the format %d.

94494 SEGSZ (a,b) The size of the associated shared memory segment. This field shall be written
94495 using the format %d.

94496 CPID (a,p) The process ID of the creator of the shared memory entry. This field shall be
94497 written using the format %d.

94498 LPID (a,p) The process ID of the last process to attach or detach the shared memory
94499 segment. This field shall be written using the format:

94500 "%d", <pid>

94501 where <pid> is 0 if no process has attached the corresponding shared memory
94502 segment; otherwise, <pid> shall be the process ID of the last process to attach
94503 or detach the segment.

94504 ATIME (a,t) The time the last attach on the associated shared memory segment was
94505 completed. If the corresponding shared memory segment has ever been
94506 attached, the hour, minute, and second of the last time the segment was
94507 attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the
94508 format " no-entry" shall be written.

94509 DTIME (a,t) The time the last detach on the associated shared memory segment was
94510 completed. If the corresponding shared memory segment has ever been
94511 detached, the hour, minute, and second of the last time the segment was
94512 detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the
94513 format " no-entry" shall be written.

94514 The following four columns shall be only written out for semaphore sets:

94515 CREATOR (a,c) The user of the creator of the facility entry. If the user name of the creator is
94516 found in the user database, at least the first eight column positions of the
94517 name shall be written using the format %s. Otherwise, the user ID of the
94518 creator shall be written using the format %d.

94519 CGROUP (a,c) The group name of the creator of the facility entry. If the group name of the
94520 creator is found in the group database, at least the first eight column positions
94521 of the name shall be written using the format %s. Otherwise, the group ID of
94522 the creator shall be written using the format %d.

- 94523 NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.
94524 This field shall be written using the format %d.
- 94525 OTIME (a,t) The time the last semaphore operation on the set associated with the
94526 semaphore entry was completed. If a semaphore operation has ever been
94527 performed on the corresponding semaphore set, the hour, minute, and second
94528 of the last semaphore operation on the semaphore set shall be written using
94529 the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be
94530 written.
- 94531 The following column shall be written for all three reports when it is requested:
- 94532 CTIME (a,t) The time the associated entry was created or changed. The hour, minute, and
94533 second of the time when the associated entry was created shall be written
94534 using the format %d:%2.2d:%2.2d.
- 94535 **STDERR**
94536 The standard error shall be used only for diagnostic messages.
- 94537 **OUTPUT FILES**
94538 None.
- 94539 **EXTENDED DESCRIPTION**
94540 None.
- 94541 **EXIT STATUS**
94542 The following exit values shall be returned:
94543 0 Successful completion.
94544 >0 An error occurred.
- 94545 **CONSEQUENCES OF ERRORS**
94546 Default.
- 94547 **APPLICATION USAGE**
94548 Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate
94549 only when it was retrieved.
- 94550 **EXAMPLES**
94551 None.
- 94552 **RATIONALE**
94553 None.
- 94554 **FUTURE DIRECTIONS**
94555 None.
- 94556 **SEE ALSO**
94557 *ipcrm*
94558 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
94559 XSH *msgrcv()*, *msgsnd()*, *semget()*, *semop()*, *shmat()*, *shmdt()*, *shmget()*
- 94560 **CHANGE HISTORY**
94561 First released in Issue 5.

94562 **Issue 6**

94563 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

94564 The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

94565 The Open Group Base Resolution bwg98-004 is applied.

94566 **Issue 7**

94567 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94568 SD5-XCU-ERN-139 is applied, adding the *ipcrm* utility to the SEE ALSO section.

94569 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0108 [584] is applied.

94570 **NAME**

94571 jobs — display status of jobs in the current session

94572 **SYNOPSIS**94573 UP `jobs [-l|-p] [job_id...]`94574 **DESCRIPTION**94575 The *jobs* utility shall display the status of jobs that were started in the current shell environment;
94576 see [Section 2.12](#) (on page 2381).94577 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the
94578 list of those “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page
94579 2370).94580 **OPTIONS**94581 The *jobs* utility shall conform to XBD [Section 12.2](#) (on page 216).

94582 The following options shall be supported:

94583 **-l** (The letter ell.) Provide more information about each job listed. This information
94584 shall include the job number, current job, process group ID, state, and the
94585 command that formed the job.94586 **-p** Display only the process IDs for the process group leaders of the selected jobs.94587 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs
94588 and all jobs whose status has changed and have not been reported by the shell.94589 **OPERANDS**

94590 The following operand shall be supported:

94591 *job_id* Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the
94592 status information for all jobs shall be displayed. The format of *job_id* is described
94593 in XBD [Section 3.204](#) (on page 66).94594 **STDIN**

94595 Not used.

94596 **INPUT FILES**

94597 None.

94598 **ENVIRONMENT VARIABLES**94599 The following environment variables shall affect the execution of *jobs*:94600 **LANG** Provide a default value for the internationalization variables that are unset or null.
94601 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
94602 variables used to determine the values of locale categories.)94603 **LC_ALL** If set to a non-empty string value, override the values of all the other
94604 internationalization variables.94605 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
94606 characters (for example, single-byte as opposed to multi-byte characters in
94607 arguments).94608 **LC_MESSAGES**94609 Determine the locale that should be used to affect the format and contents of
94610 diagnostic messages written to standard error and informative messages written to
94611 standard output.

94612 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

94613 **ASYNCHRONOUS EVENTS**

94614 Default.

94615 **STDOUT**

94616 If the **-p** option is specified, the output shall consist of one line for each process ID:

94617 "%d\n", <process ID>

94618 Otherwise, if the **-l** option is not specified, the output shall be a series of lines of the form:

94619 "[%d] %c %s %s\n", <job-number>, <current>, <state>, <command>

94620 where the fields shall be as follows:

94621 <current> The character '+' identifies the job that would be used as a default for the *fg* or *bg*

94622 utilities; this job can also be specified using the *job_id* %+ or "%%". The character

94623 '-' identifies the job that would become the default if the current default job were

94624 to exit; this job can also be specified using the *job_id* %-. For other jobs, this field is

94625 a <space>. At most one job can be identified with '+' and at most one job can be

94626 identified with '-'. If there is any suspended job, then the current job shall be a

94627 suspended job. If there are at least two suspended jobs, then the previous job also

94628 shall be a suspended job.

94629 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*

94630 utilities. Using these utilities, the job can be identified by prefixing the job number

94631 with '% '.

94632 <state> One of the following strings (in the POSIX locale):

94633 **Running** Indicates that the job has not been suspended by a signal and has not

94634 exited.

94635 **Done** Indicates that the job completed and returned exit status zero.

94636 **Done(code)** Indicates that the job completed normally and that it exited with the

94637 specified non-zero exit status, *code*, expressed as a decimal number.

94638 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.

94639 **Stopped (SIGTSTP)**

94640 Indicates that the job was suspended by the SIGTSTP signal.

94641 **Stopped (SIGSTOP)**

94642 Indicates that the job was suspended by the SIGSTOP signal.

94643 **Stopped (SIGTTIN)**

94644 Indicates that the job was suspended by the SIGTTIN signal.

94645 **Stopped (SIGTTOU)**

94646 Indicates that the job was suspended by the SIGTTOU signal.

94647 The implementation may substitute the string **Suspended** in place of **Stopped**. If

94648 the job was terminated by a signal, the format of <state> is unspecified, but it shall

94649 be visibly distinct from all of the other <state> formats shown here and shall

94650 indicate the name or description of the signal causing the termination.

94651 <command> The associated command that was given to the shell.

94652 If the **-l** option is specified, a field containing the process group ID shall be inserted before the

94653 <state> field. Also, more processes in a process group may be output on separate lines, using

94654 only the process ID and *<command>* fields.

94655 **STDERR**

94656 The standard error shall be used only for diagnostic messages.

94657 **OUTPUT FILES**

94658 None.

94659 **EXTENDED DESCRIPTION**

94660 None.

94661 **EXIT STATUS**

94662 The following exit values shall be returned:

94663 0 Successful completion.

94664 >0 An error occurred.

94665 **CONSEQUENCES OF ERRORS**

94666 Default.

94667 **APPLICATION USAGE**

94668 The *-p* option is the only portable way to find out the process group of a job because different implementations have different strategies for defining the process group of the job. Usage such as *\$(jobs -p)* provides a way of referring to the process group of the job in an implementation-independent way.

94672 The *jobs* utility does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg*. For this reason, *jobs* is generally implemented as a shell regular built-in.

94676 **EXAMPLES**

94677 None.

94678 **RATIONALE**

94679 Both "%%" and "%+" are used to refer to the current job. Both forms are of equal validity ‡the
94680 "%%" mirroring "\$\$" and "%+" mirroring the output of *jobs*. Both forms reflect historical
94681 practice of the KornShell and the C shell with job control.

94682 The job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard
94683 developers examined the characteristics of the C shell versions of these utilities and found that
94684 differences exist. Despite widespread use of the C shell, the KornShell versions were selected for
94685 this volume of POSIX.1-2017 to maintain a degree of uniformity with the rest of the KornShell
94686 features selected (such as the very popular command line editing features).

94687 The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg*
94688 utilities because *jobs* is useful for examining background jobs, regardless of the condition of job
94689 control. When the user has invoked a *set +m* command and job control has been turned off, *jobs*
94690 can still be used to examine the background jobs associated with that current session. Similarly,
94691 *kill* can then be used to kill background jobs with *kill %<background job number>*.

94692 The output for terminated jobs is left unspecified to accommodate various historical systems.
94693 The following formats have been witnessed:

- 94694 1. **Killed**(*signal name*)
- 94695 2. *signal name*

94696 3. *signal name*(**coredump**)

94697 4. *signal description*– **core dumped**

94698 Most users should be able to understand these formats, although it means that applications have
94699 trouble parsing them.

94700 The calculation of job IDs was not described since this would suggest an implementation, which
94701 may impose unnecessary restrictions.

94702 In an early proposal, a **-n** option was included to “Display the status of jobs that have changed,
94703 exited, or stopped since the last status report”. It was removed because the shell always writes
94704 any changed status of jobs before each prompt.

94705 **FUTURE DIRECTIONS**

94706 None.

94707 **SEE ALSO**

94708 [Section 2.12](#) (on page 2381), *bg*, *fg*, *kill*, *wait*

94709 [XBD Section 3.204](#) (on page 66), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

94710 **CHANGE HISTORY**

94711 First released in Issue 4.

94712 **Issue 6**

94713 This utility is marked as part of the User Portability Utilities option.

94714 The JC shading is removed as job control is mandatory in this version.

94715 **Issue 7**

94716 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

94717 **NAME**

94718 join — relational database operator

94719 **SYNOPSIS**94720 join **[-a file_number|-v file_number]** **[-e string]** **[-o list]** **[-t char]**
94721 **[-1 field]** **[-2 field]** file1 file294722 **DESCRIPTION**94723 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be
94724 written to the standard output.94725 The join field is a field in each file on which the files are compared. The *join* utility shall write
94726 one line in the output for each pair of lines in *file1* and *file2* that have join fields that collate
94727 equally. The output line by default shall consist of the join field, then the remaining fields from
94728 *file1*, then the remaining fields from *file2*. This format can be changed by using the **-o** option
94729 (see below). The **-a** option can be used to add unmatched lines to the output. The **-v** option can
94730 be used to output only unmatched lines.94731 The files *file1* and *file2* shall be ordered in the collating sequence of *sort -b* on the fields on which
94732 they shall be joined, by default the first in each line. All selected output shall be written in the
94733 same collating sequence.94734 The default input field separators shall be <blank> characters. In this case, multiple separators
94735 shall count as one field separator, and leading separators shall be ignored. The default output
94736 field separator shall be a <space>.94737 The field separator and collating sequence can be changed by using the **-t** option (see below).94738 If the same key appears more than once in either file, all combinations of the set of remaining
94739 fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines
94740 encountered.

94741 If the input files are not in the appropriate collating sequence, the results are unspecified.

94742 **OPTIONS**94743 The *join* utility shall conform to XBD [Section 12.2](#) (on page 216).

94744 The following options shall be supported:

94745 **-a file_number**94746 Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or
94747 2, in addition to the default output. If both **-a1** and **-a2** are specified, all unpairable
94748 lines shall be output.94749 **-e string** Replace empty output fields in the list selected by **-o** with the string *string*.94750 **-o list** Construct the output line to comprise the fields specified in *list*, each element of
94751 which shall have one of the following two forms:94752 1. *file_number.field*, where *file_number* is a file number and *field* is a decimal
94753 integer field number

94754 2. 0 (zero), representing the join field

94755 The elements of *list* shall be either <comma>-separated or <blank>-separated, as
94756 specified in Guideline 8 of XBD [Section 12.2](#) (on page 216). The fields specified by
94757 *list* shall be written for all selected output lines. Fields selected by *list* that do not
94758 appear in the input shall be treated as empty output fields. (See the **-e** option.)
94759 Only specifically requested fields shall be written. The application shall ensure that
94760 *list* is a single command line argument.

- 94761 **-t char** Use character *char* as a separator, for both input and output. Every appearance of *char* in a line shall be significant. When this option is specified, the collating sequence shall be the same as *sort* without the **-b** option.
- 94762
- 94763
- 94764 **-v file_number**
- 94765 Instead of the default output, produce a line only for each unpairable line in *file_number*, where *file_number* is 1 or 2. If both **-v1** and **-v2** are specified, all unpairable lines shall be output.
- 94766
- 94767
- 94768 **-1 field** Join on the *field*th field of file 1. Fields are decimal integers starting with 1.
- 94769 **-2 field** Join on the *field*th field of file 2. Fields are decimal integers starting with 1.
- 94770 **OPERANDS**
- 94771 The following operands shall be supported:
- 94772 *file1, file2* A pathname of a file to be joined. If either of the *file1* or *file2* operands is '-', the standard input shall be used in its place.
- 94773
- 94774 **STDIN**
- 94775 The standard input shall be used only if the *file1* or *file2* operand is '-'. See the INPUT FILES section.
- 94776
- 94777 **INPUT FILES**
- 94778 The input files shall be text files.
- 94779 **ENVIRONMENT VARIABLES**
- 94780 The following environment variables shall affect the execution of *join*:
- 94781 **LANG** Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)
- 94782
- 94783
- 94784 **LC_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.
- 94785
- 94786 **LC_COLLATE**
- 94787 Determine the locale of the collating sequence *join* expects to have been used when the input files were sorted.
- 94788
- 94789 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
- 94790
- 94791
- 94792 **LC_MESSAGES**
- 94793 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 94794
- 94795 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 94796 **ASYNCHRONOUS EVENTS**
- 94797 Default.
- 94798 **STDOUT**
- 94799 The *join* utility output shall be a concatenation of selected character fields. When the **-o** option is not specified, the output shall be:
- 94800
- 94801 "%s%s%s\n", <*join field*>, <*other file1 fields*>,
94802 <*other file2 fields*>
- 94803 If the join field is not the first field in a file, the <*other file fields*> for that file shall be:

94804 `<fields preceding join field>, <fields following join field>`

94805 When the `-o` option is specified, the output format shall be:

94806 `"%s\n", <concatenation of fields>`

94807 where the concatenation of fields is described by the `-o` option, above.

94808 For either format, each field (except the last) shall be written with its trailing separator character.
 94809 If the separator is the default (`<blank>` characters), a single `<space>` shall be written after each
 94810 field (except the last).

94811 **STDERR**

94812 The standard error shall be used only for diagnostic messages.

94813 **OUTPUT FILES**

94814 None.

94815 **EXTENDED DESCRIPTION**

94816 None.

94817 **EXIT STATUS**

94818 The following exit values shall be returned:

94819 0 All input files were output successfully.

94820 >0 An error occurred.

94821 **CONSEQUENCES OF ERRORS**

94822 Default.

94823 **APPLICATION USAGE**

94824 Pathnames consisting of numeric digits or of the form *string.string* should not be specified
 94825 directly following the `-o` list.

94826 If the collating sequence of the current locale does not have a total ordering of all characters (see
 94827 XBD Section 7.3.2, on page 147), *join* treats fields that collate equally but are not identical as
 94828 being the same. If this behavior is not desired, it can be avoided by forcing the use of the POSIX
 94829 locale (although this means re-sorting the input files into the POSIX locale collating sequence.)

94830 When using *join* to process pathnames, it is recommended that `LC_ALL`, or at least `LC_CTYPE`
 94831 and `LC_COLLATE`, are set to `POSIX` or `C` in the environment, since pathnames can contain byte
 94832 sequences that do not form valid characters in some locales, in which case the utility's behavior
 94833 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore
 94834 this problem is avoided.

94835 **EXAMPLES**

94836 The `-o 0` field essentially selects the union of the join fields. For example, given file **phone**:

```
94837 !Name           Phone Number
94838 Don             +1 123-456-7890
94839 Hal            +1 234-567-8901
94840 Yasushi        +2 345-678-9012
```

94841 and file **fax**:

```
94842 !Name           Fax Number
94843 Don             +1 123-456-7899
94844 Keith          +1 456-789-0122
94845 Yasushi        +2 345-678-9011
```

94846 (where the large expanses of white space are meant to each represent a single <tab>), the
94847 command:

```
94848 join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax
```

94849 (where <tab> is a literal <tab> character) would produce:

```
94850 !Name           Phone Number           Fax Number
94851 Don             +1 123-456-7890        +1 123-456-7899
94852 Hal            +1 234-567-8901        (unknown)
94853 Keith          (unknown)               +1 456-789-0122
94854 Yasushi        +2 345-678-9012        +2 345-678-9011
```

94855 Multiple instances of the same key will produce combinatorial results. The following:

```
94856 fa:
94857     a x
94858     a y
94859     a z
94860 fb:
94861     a p
```

94862 will produce:

```
94863 a x p
94864 a y p
94865 a z p
```

94866 And the following:

```
94867 fa:
94868     a b c
94869     a d e
94870 fb:
94871     a w x
94872     a y z
94873     a o p
```

94874 will produce:

```
94875 a b c w x
94876 a b c y z
94877 a b c o p
94878 a d e w x
94879 a d e y z
94880 a d e o p
```

94881 RATIONALE

94882 The `-e` option is only effective when used with `-o` because, unless specific fields are identified
94883 using `-o`, *join* is not aware of what fields might be empty. The exception to this is the join field,
94884 but identifying an empty join field with the `-e` string is not historical practice and some scripts
94885 might break if this were changed.

94886 The 0 field in the `-o` list was adopted from the Tenth Edition version of *join* to satisfy
94887 international objections that the *join* in the base documents for IEEE Std 1003.2-1992 did not
94888 support the “full join” or “outer join” described in relational database literature. Although it has
94889 been possible to include a join field in the output (by default, or by field number using `-o`), the
94890 join field could not be included for an unpaired line selected by `-a`. The `-o 0` field essentially

- 94891 selects the union of the join fields.
- 94892 This sort of outer join was not possible with the *join* commands in the base documents for
94893 IEEE Std 1003.2-1992. The `-o 0` field was chosen because it is an upwards-compatible change for
94894 applications. An alternative was considered: have the join field represent the union of the fields
94895 in the files (where they are identical for matched lines, and one or both are null for unmatched
94896 lines). This was not adopted because it would break some historical applications.
- 94897 The ability to specify *file2* as `-` is not historical practice; it was added for completeness.
- 94898 The `-v` option is not historical practice, but was considered necessary because it permitted the
94899 writing of *only* those lines that do not match on the join field, as opposed to the `-a` option, which
94900 prints both lines that do and do not match. This additional facility is parallel with the `-v` option
94901 of *grep*.
- 94902 Some historical implementations have been encountered where a blank line in one of the input
94903 files was considered to be the end of the file; the description in this volume of POSIX.1-2017 does
94904 not cite this as an allowable case.
- 94905 Earlier versions of this standard allowed `-j`, `-j1`, `-j2` options, and a form of the `-o` option that
94906 allowed the *list* option-argument to be multiple arguments. These forms are no longer specified
94907 by POSIX.1-2017 but may be present in some implementations.
- 94908 **FUTURE DIRECTIONS**
- 94909 None.
- 94910 **SEE ALSO**
- 94911 [awk](#), [comm](#), [sort](#), [uniq](#)
- 94912 XBD [Section 7.3.2](#) (on page 147), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 94913 **CHANGE HISTORY**
- 94914 First released in Issue 2.
- 94915 **Issue 6**
- 94916 The obsolescent `-j` options and the multi-argument `-o` option are removed in this version.
- 94917 The normative text is reworded to avoid use of the term “must” for application requirements.
- 94918 **Issue 7**
- 94919 Austin Group Interpretation 1003.1-2001 #027 is applied.
- 94920 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 94921 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0109 [963], XCU/TC2-2008/0110 [663],
94922 XCU/TC2-2008/0111 [971], and XCU/TC2-2008/0112 [885] are applied.

94923 **NAME**94924 `kill` — terminate or signal processes94925 **SYNOPSIS**94926 `kill -s signal_name pid...`94927 `kill -l [exit_status]`94928 XSI `kill [-signal_name] pid...`94929 `kill [-signal_number] pid...`94930 **DESCRIPTION**94931 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.94932 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function
94933 defined in the System Interfaces volume of POSIX.1-2017 called with the following arguments:94934 The value of the *pid* operand shall be used as the *pid* argument.94935 The *sig* argument is the value specified by the `-s` option, `-signal_number` option, or the
94936 `-signal_name` option, or by SIGTERM, if none of these options is specified.94937 **OPTIONS**94938 XSI The *kill* utility shall conform to XBD [Section 12.2](#) (on page 216), except that in the last two
94939 SYNOPSIS forms, the `-signal_number` and `-signal_name` options are usually more than a single
94940 character.

94941 The following options shall be supported:

94942 `-l` (The letter ell.) Write all values of *signal_name* supported by the implementation, if
94943 no operand is given. If an *exit_status* operand is given and it is a value of the '?'
94944 shell special parameter (see [Section 2.5.2](#) (on page 2350) and *wait*) corresponding to
94945 a process that was terminated by a signal, the *signal_name* corresponding to the
94946 signal that terminated the process shall be written. If an *exit_status* operand is
94947 given and it is the unsigned decimal integer value of a signal number, the
94948 *signal_name* (the symbolic constant name without the **SIG** prefix defined in the
94949 Base Definitions volume of POSIX.1-2017) corresponding to that signal shall be
94950 written. Otherwise, the results are unspecified.94951 `-s signal_name`94952 Specify the signal to send, using one of the symbolic names defined in the
94953 `<signal.h>` header. Values of *signal_name* shall be recognized in a case-independent
94954 fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be
94955 recognized, representing the signal value zero. The corresponding signal shall be
94956 sent instead of SIGTERM.94957 XSI `-signal_name`94958 Equivalent to `-s signal_name`.94959 XSI `-signal_number`94960 Specify a non-negative decimal integer, *signal_number*, representing the signal to be
94961 used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The
94962 correspondence between integer values and the *sig* value used is shown in the
94963 following list.94964 The effects of specifying any *signal_number* other than those listed below are
94965 undefined.

94966	0	0
94967	1	SIGHUP
94968	2	SIGINT
94969	3	SIGQUIT
94970	6	SIGABRT
94971	9	SIGKILL
94972	14	SIGALRM
94973	15	SIGTERM
94974	If the first argument is a negative integer, it shall be interpreted as a <i>-signal_number</i>	
94975	option, not as a negative <i>pid</i> operand specifying a process group.	

OPERANDS

94976 The following operands shall be supported:

94977 *pid* One of the following:

- 94979 1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative, and zero values of the *pid* operand shall be as described for the *kill()* function. If process number 0 is specified, all processes in the current process group shall be signaled. For the effects of negative *pid* numbers, see the *kill()* function defined in the System Interfaces volume of POSIX.1-2017. If the first *pid* operand is negative, it should be preceded by "--" to keep it from being interpreted as an option.
- 94980 2. A job control job ID (see XBD [Section 3.204](#), on page 66) that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of *kill* in the current shell execution environment; see [Section 2.12](#) (on page 2381).

94991 *exit_status* A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

STDIN

94994 Not used.

INPUT FILES

94996 None.

ENVIRONMENT VARIABLES

94998 The following environment variables shall affect the execution of *kill*:

94999 *LANG* Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization variables used to determine the values of locale categories.)

95002 *LC_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.

95004 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

95007 **LC_MESSAGES**
 95008 Determine the locale that should be used to affect the format and contents of
 95009 diagnostic messages written to standard error.

95010 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

95011 **ASYNCHRONOUS EVENTS**
 95012 Default.

95013 **STDOUT**
 95014 When the `-l` option is not specified, the standard output shall not be used.
 95015 When the `-l` option is specified, the symbolic name of each signal shall be written in the
 95016 following format:
 95017 `"%s%c", <signal_name>, <separator>`
 95018 where the *<signal_name>* is in uppercase, without the **SIG** prefix, and the *<separator>* shall be
 95019 either a *<newline>* or a *<space>*. For the last signal written, *<separator>* shall be a *<newline>*.
 95020 When both the `-l` option and *exit_status* operand are specified, the symbolic name of the
 95021 corresponding signal shall be written in the following format:
 95022 `"%s\n", <signal_name>`

95023 **STDERR**
 95024 The standard error shall be used only for diagnostic messages.

95025 **OUTPUT FILES**
 95026 None.

95027 **EXTENDED DESCRIPTION**
 95028 None.

95029 **EXIT STATUS**
 95030 The following exit values shall be returned:
 95031 0 At least one matching process was found for each *pid* operand, and the specified signal was
 95032 successfully processed for at least one matching process.
 95033 >0 An error occurred.

95034 **CONSEQUENCES OF ERRORS**
 95035 Default.

95036 **APPLICATION USAGE**
 95037 Process numbers can be found by using *ps*.
 95038 The job control job ID notation is not required to work as expected when *kill* is operating in its
 95039 own utility execution environment. In either of the following examples:
 95040 `nohup kill %1 &`
 95041 `system("kill %1");`
 95042 the *kill* operates in a different environment and does not share the shell's understanding of job
 95043 numbers.

95044 **EXAMPLES**
 95045 Any of the commands:
 95046 `kill -9 100 -165`
 95047 `kill -s kill 100 -165`


```
95048 kill -s KILL 100 -165
```

95049 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose
95050 process group ID is 165, assuming the sending process has permission to send that signal to the
95051 specified processes, and that they exist.

95052 The System Interfaces volume of POSIX.1-2017 and this volume of POSIX.1-2017 do not require
95053 specific signal numbers for any *signal_names*. Even the *-signal_number* option provides symbolic
95054 (although numeric) names for signals. If a process is terminated by a signal, its exit status
95055 indicates the signal that killed it, but the exact values are not specified. The *kill -l* option,
95056 however, can be used to map decimal signal numbers and exit status values into the name of a
95057 signal. The following example reports the status of a terminated job:

```
95058 job
95059 stat=$?
95060 if [ $stat -eq 0 ]
95061 then
95062     echo job completed successfully.
95063 elif [ $stat -gt 128 ]
95064 then
95065     echo job terminated by signal SIG$(kill -l $stat).
95066 else
95067     echo job terminated with error code $stat.
95068 fi
```

95069 To send the default signal to a process group (say 123), an application should use a command
95070 similar to one of the following:

```
95071 kill -TERM -123
95072 kill -- -123
```

95073 RATIONALE

95074 The *-l* option originated from the C shell, and is also implemented in the KornShell. The C shell
95075 output can consist of multiple output lines because the signal names do not always fit on a
95076 single line on some terminal screens. The KornShell output also included the implementation-
95077 defined signal numbers and was considered by the standard developers to be too difficult for
95078 scripts to parse conveniently. The specified output format is intended not only to accommodate
95079 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing
95080 on systems for which this is appropriate.

95081 An early proposal invented the name SIGNULL as a *signal_name* for signal 0 (used by the System
95082 Interfaces volume of POSIX.1-2017 to test for the existence of a process without sending it a
95083 signal). Since the *signal_name* 0 can be used in this case unambiguously, SIGNULL has been
95084 removed.

95085 An early proposal also required symbolic *signal_names* to be recognized with or without the **SIG**
95086 prefix. Historical versions of *kill* have not written the **SIG** prefix for the *-l* option and have not
95087 recognized the **SIG** prefix on *signal_names*. Since neither applications portability nor ease-of-use
95088 would be improved by requiring this extension, it is no longer required.

95089 To avoid an ambiguity of an initial negative number argument specifying either a signal number
95090 or a process group, POSIX.1-2017 mandates that it is always considered the former by
95091 implementations that support the XSI option. It also requires that conforming applications
95092 always use the "--" options terminator argument when specifying a process group, unless an
95093 option is also specified.

95094 The *-s* option was added in response to international interest in providing some form of *kill* that

95095 meets the Utility Syntax Guidelines.

95096 The job control job ID notation is not required to work as expected when *kill* is operating in its
95097 own utility execution environment. In either of the following examples:

```
95098 nohup kill %1 &  
95099 system("kill %1");
```

95100 the *kill* operates in a different environment and does not understand how the shell has managed
95101 its job numbers.

95102 **FUTURE DIRECTIONS**

95103 None.

95104 **SEE ALSO**

95105 [Chapter 2](#) (on page 2345), *ps*, *wait*

95106 [XBD Section 3.204](#) (on page 66), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216), [<signal.h>](#)

95107 XSH *kill()*

95108 **CHANGE HISTORY**

95109 First released in Issue 2.

95110 **Issue 6**

95111 The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI
95112 option, corresponding to a similar change in the *trap* special built-in.

95113 **Issue 7**

95114 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

95115 **NAME**95116 lex — generate programs for lexical tasks (**DEVELOPMENT**)95117 **SYNOPSIS**95118 CD `lex [-t] [-n|-v] [file...]`95119 **DESCRIPTION**

95120 The *lex* utility shall generate C programs to be used in lexical processing of character input, and
 95121 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code
 95122 and conform to the ISO C standard, without depending on any undefined, unspecified, or
 95123 implementation-defined behavior, except in cases where the code is copied directly from the
 95124 supplied source, or in cases that are documented by the implementation. Usually, the *lex* utility
 95125 shall write the program it generates to the file `lex.yy.c`; the state of this file is unspecified if *lex*
 95126 exits with a non-zero exit status. See the EXTENDED DESCRIPTION section for a complete
 95127 description of the *lex* input language.

95128 **OPTIONS**95129 The *lex* utility shall conform to XBD [Section 12.2](#) (on page 216), except for Guideline 9.

95130 The following options shall be supported:

- 95131 **-n** Suppress the summary of statistics usually written with the `-v` option. If no table
 95132 sizes are specified in the *lex* source code and the `-v` option is not specified, then `-n`
 95133 is implied.
- 95134 **-t** Write the resulting program to standard output instead of `lex.yy.c`.
- 95135 **-v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*
 95136 table sizes in [Definitions in lex](#) (on page 2887).) If the `-t` option is specified and `-n`
 95137 is not specified, this report shall be written to standard error. If table sizes are
 95138 specified in the *lex* source code, and if the `-n` option is not specified, the `-v` option
 95139 may be enabled.

95140 **OPERANDS**

95141 The following operand shall be supported:

- 95142 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be
 95143 concatenated to produce a single *lex* program. If no *file* operands are specified, or if
 95144 a *file* operand is `'-'`, the standard input shall be used.

95145 **STDIN**

95146 The standard input shall be used if no *file* operands are specified, or if a *file* operand is `'-'`. See
 95147 INPUT FILES.

95148 **INPUT FILES**

95149 The input files shall be text files containing *lex* source code, as described in the EXTENDED
 95150 DESCRIPTION section.

95151 **ENVIRONMENT VARIABLES**95152 The following environment variables shall affect the execution of *lex*:

- 95153 **LANG** Provide a default value for the internationalization variables that are unset or null.
 95154 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 95155 variables used to determine the values of locale categories.)
- 95156 **LC_ALL** If set to a non-empty string value, override the values of all the other
 95157 internationalization variables.

- 95158 *LC_COLLATE*
- 95159 Determine the locale for the behavior of ranges, equivalence classes, and multi-
- 95160 character collating elements within regular expressions. If this variable is not set to
- 95161 the POSIX locale, the results are unspecified.
- 95162 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 95163 characters (for example, single-byte as opposed to multi-byte characters in
- 95164 arguments and input files), and the behavior of character classes within regular
- 95165 expressions. If this variable is not set to the POSIX locale, the results are
- 95166 unspecified.
- 95167 *LC_MESSAGES*
- 95168 Determine the locale that should be used to affect the format and contents of
- 95169 diagnostic messages written to standard error.
- 95170 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 95171 **ASYNCHRONOUS EVENTS**
- 95172 Default.
- 95173 **STDOUT**
- 95174 If the `-t` option is specified, the text file of C source code output of *lex* shall be written to
- 95175 standard output.
- 95176 If the `-t` option is not specified:
- 95177 Implementation-defined informational, error, and warning messages concerning the
- 95178 contents of *lex* source code input shall be written to either the standard output or standard
- 95179 error.
- 95180 If the `-v` option is specified and the `-n` option is not specified, *lex* statistics shall also be
- 95181 written to either the standard output or standard error, in an implementation-defined
- 95182 format. These statistics may also be generated if table sizes are specified with a '%'
- 95183 operator in the *Definitions* section, as long as the `-n` option is not specified.
- 95184 **STDERR**
- 95185 If the `-t` option is specified, implementation-defined informational, error, and warning messages
- 95186 concerning the contents of *lex* source code input shall be written to the standard error.
- 95187 If the `-t` option is not specified:
- 95188 1. Implementation-defined informational, error, and warning messages concerning the
- 95189 contents of *lex* source code input shall be written to either the standard output or
- 95190 standard error.
- 95191 2. If the `-v` option is specified and the `-n` option is not specified, *lex* statistics shall also be
- 95192 written to either the standard output or standard error, in an implementation-defined
- 95193 format. These statistics may also be generated if table sizes are specified with a '%'
- 95194 operator in the *Definitions* section, as long as the `-n` option is not specified.
- 95195 **OUTPUT FILES**
- 95196 A text file containing C source code shall be written to *lex.yy.c*, or to the standard output if the `-t`
- 95197 option is present.
- 95198 **EXTENDED DESCRIPTION**
- 95199 Each input file shall contain *lex* source code, which is a table of regular expressions with
- 95200 corresponding actions in the form of C program fragments.
- 95201 When *lex.yy.c* is compiled and linked with the *lex* library (using the `-ll` operand with *c99*), the

95202 resulting program shall read character input from the standard input and shall partition it into
95203 strings that match the given expressions.

95204 When an expression is matched, these actions shall occur:

95205 The input string that was matched shall be left in *ytext* as a null-terminated string; *ytext*
95206 shall either be an external character array or a pointer to a character string. As explained in
95207 [Definitions in lex](#) (on page 2887), the type can be explicitly selected using the **%array** or
95208 **%pointer** declarations, but the default is implementation-defined.

95209 The external **int** *yyleng* shall be set to the length of the matching string.

95210 The expression's corresponding program fragment, or action, shall be executed.

95211 During pattern matching, *lex* shall search the set of patterns for the single longest possible
95212 match. Among rules that match the same number of characters, the rule given first shall be
95213 chosen.

95214 The general format of *lex* source shall be:

```
95215     Definitions
95216     %%
95217     Rules
95218     %%
95219     UserSubroutines
```

95220 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);
95221 the second "%%" is required only if user subroutines follow.

95222 Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program
95223 fragment and shall be copied to the external definition area of the **lex.yy.c** file. Similarly,
95224 anything in the *Definitions* section included between delimiter lines containing only "%{" and
95225 "%}" shall also be copied unchanged to the external definition area of the **lex.yy.c** file.

95226 Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing
95227 at the beginning of the *Rules* section before any rules are specified shall be written to **lex.yy.c**
95228 after the declarations of variables for the *yylex()* function and before the first line of code in
95229 *yylex()*. Thus, user variables local to *yylex()* can be declared here, as well as application code to
95230 execute upon entry to *yylex()*.

95231 The action taken by *lex* when encountering any input beginning with a <blank> or within "%{"
95232 and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is
95233 undefined. The presence of such input may result in an erroneous definition of the *yylex()*
95234 function.

95235 C-language code in the input shall not contain C-language trigraphs. The C-language code
95236 within "%{" and "%}" delimiter lines shall not contain any lines consisting only of "%}", or
95237 only of "%%".

95238 **Definitions in lex**

95239 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between
 95240 "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex*
 95241 substitution string. The format of these lines shall be:

95242 *name substitute*

95243 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is
 95244 undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name*
 95245 string shall be recognized in this context only when the braces are provided and when it does
 95246 not appear within a bracket expression or within double-quotes.

95247 In the *Definitions* section, any line beginning with a <percent-sign> ('%') character and followed
 95248 by an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.
 95249 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall
 95250 define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns
 95251 with no state specified shall be also active; in a %x state, such patterns shall not be active. The
 95252 rest of the line, after the first word, shall be considered to be one or more <blank>-separated
 95253 names of start conditions. Start condition names shall be constructed in the same way as
 95254 definition names. Start conditions can be used to restrict the matching of regular expressions to
 95255 one or more states as described in [Regular Expressions in lex](#) (on page 2888).

95256 Implementations shall accept either of the following two mutually-exclusive declarations in the
 95257 *Definitions* section:

95258 **%array** Declare the type of *yytext* to be a null-terminated character array.

95259 **%pointer** Declare the type of *yytext* to be a pointer to a null-terminated character string.

95260 The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of
 95261 the scanner source file (that is, via an **extern**), the application shall include the appropriate
 95262 **%array** or **%pointer** declaration in the scanner source file.

95263 Implementations shall accept declarations in the *Definitions* section for setting certain internal
 95264 table sizes. The declarations are shown in the following table.

95265 **Table 4-10** Table Size Declarations in *lex*

Declaration	Description	Minimum Value
%p <i>n</i>	Number of positions	2 500
%n <i>n</i>	Number of states	500
%a <i>n</i>	Number of transitions	2 000
%e <i>n</i>	Number of parse tree nodes	1 000
%k <i>n</i>	Number of packed character classes	1 000
%o <i>n</i>	Size of the output array	3 000

95273 In the table, *n* represents a positive decimal integer, preceded by one or more <blank>
 95274 characters. The exact meaning of these table size numbers is implementation-defined. The
 95275 implementation shall document how these numbers affect the *lex* utility and how they are
 95276 related to any output that may be generated by the implementation should limitations be
 95277 encountered during the execution of *lex*. It shall be possible to determine from this output
 95278 which of the table size values needs to be modified to permit *lex* to successfully generate tables
 95279 for the input language. The values in the column Minimum Value represent the lowest values
 95280 conforming implementations shall provide.

95281 **Rules in lex**

95282 The rules in *lex* source files are a table in which the left column contains regular expressions and
 95283 the right column contains actions (C program fragments) to be executed when the expressions
 95284 are recognized.

95285 *ERE action*

95286 *ERE action*

95287 . . .

95288 The extended regular expression (ERE) portion of a row shall be separated from *action* by one or
 95289 more <blank> characters. A regular expression containing <blank> characters shall be
 95290 recognized under one of the following conditions:

95291 The entire expression appears within double-quotes.

95292 The <blank> characters appear within double-quotes or square brackets.

95293 Each <blank> is preceded by a <backslash> character.

95294 **User Subroutines in lex**

95295 Anything in the user subroutines section shall be copied to **lex.yy.c** following *yylex()*.

95296 **Regular Expressions in lex**

95297 The *lex* utility shall support the set of extended regular expressions (see XBD [Section 9.4](#), on page
 95298 188), with the following additions and exceptions to the syntax:

95299 ". . ." Any string enclosed in double-quotes shall represent the characters within the
 95300 double-quotes as themselves, except that <backslash>-escapes (which appear in
 95301 the following table) shall be recognized. Any <backslash>-escape sequence shall be
 95302 terminated by the closing quote. For example, "\01"1" represents a single
 95303 string: the octal value 1 followed by the character '1'.

95304 <state>*r*, <state1, state2, . . .>*r*

95305 The regular expression *r* shall be matched only when the program is in one of the
 95306 start conditions indicated by *state*, *state1*, and so on; see [Actions in lex](#) (on page
 95307 2890). (As an exception to the typographical conventions of the rest of this volume
 95308 of POSIX.1-2017, in this case <state> does not represent a metavariable, but the
 95309 literal angle-bracket characters surrounding a symbol.) The start condition shall be
 95310 recognized as such only at the beginning of a regular expression.

95311 *r/x* The regular expression *r* shall be matched only if it is followed by an occurrence of
 95312 regular expression *x* (*x* is the instance of trailing context, further defined below).
 95313 The token returned in *ytext* shall only match *r*. If the trailing portion of *r* matches
 95314 the beginning of *x*, the result is unspecified. The *r* expression cannot include
 95315 further trailing context or the '\$' (match-end-of-line) operator; *x* cannot include
 95316 the '^' (match-beginning-of-line) operator, nor trailing context, nor the '\$'
 95317 operator. That is, only one occurrence of trailing context is allowed in a *lex* regular
 95318 expression, and the '^' operator only can be used at the beginning of such an
 95319 expression.

95320 {*name*} When *name* is one of the substitution symbols from the *Definitions* section, the
 95321 string, including the enclosing braces, shall be replaced by the *substitute* value. The
 95322 *substitute* value shall be treated in the extended regular expression as if it were
 95323 enclosed in parentheses. No substitution shall occur if {*name*} occurs within a
 95324 bracket expression or within double-quotes.

95325 Within an ERE, a <backslash> character shall be considered to begin an escape sequence as
 95326 specified in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r',
 95327 '\t', '\v'). In addition, the escape sequences in the following table shall be recognized.

95328 A literal <newline> cannot occur within an ERE; the escape sequence '\n' can be used to
 95329 represent a <newline>. A <newline> shall not be matched by a period operator.

95330 **Table 4-11** Escape Sequences in *lex*

Escape Sequence	Description	Meaning
\digits	A <backslash> character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
\xdigits	A <backslash> character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.
\c	A <backslash> character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').	The character 'c', unchanged.

95352 **Note:** If a '\x' sequence needs to be immediately followed by a hexadecimal digit character, a
 95353 sequence such as "\x1"1" can be used, which represents a character containing the value 1,
 95354 followed by the character '1'.

95355 The order of precedence given to extended regular expressions for *lex* differs from that specified
 95356 in XBD Section 9.4 (on page 188). The order of precedence for *lex* shall be as shown in the
 95357 following table, from high to low.

95358 **Note:** The escaped characters entry is not meant to imply that these are operators, but they are
 95359 included in the table to show their relationships to the true operators. The start condition,
 95360 trailing context, and anchoring notations have been omitted from the table because of the
 95361 placement restrictions described in this section; they can only appear at the beginning or ending
 95362 of an ERE.

Table 4-12 ERE Precedence in *lex*

Extended Regular Expression	Precedence
<i>collation-related bracket symbols</i>	[= =] [: :] [. .]
<i>escaped characters</i>	\<special character>
<i>bracket expression</i>	[]
<i>quoting</i>	" . . . "
<i>grouping</i>	()
<i>definition</i>	{ name }
<i>single-character RE duplication</i>	* + ?
<i>concatenation</i>	
<i>interval expression</i>	{ m , n }
<i>alternation</i>	

The ERE anchoring operators '^' and '\$' do not appear in the table. With *lex* regular expressions, these operators are restricted in their use: the '^' operator can only be used at the beginning of an entire regular expression, and the '\$' operator only at the end. The operators apply to the entire regular expression. Thus, for example, the pattern "(^abc)|(def\$)" is undefined; it can instead be written as two separate rules, one with the regular expression "^abc" and one with "def\$", which share a common action via the special '|' action (see below). If the pattern were written "^abc|def\$", it would match either "abc" or "def" on a line by itself.

Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex* implementations. An example of embedded anchoring would be for patterns such as "(^|)foo(|\$)" to match "foo" when it exists as a complete word. This functionality can be obtained using existing *lex* features:

```
^foo/[ \n]      |
" foo"/[ \n]    /* Found foo as a separate word. */
```

Note also that '\$' is a form of trailing context (it is equivalent to "/\n") and as such cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context).

The additional regular expressions trailing-context operator '/' can be used as an ordinary character if presented within double-quotes, "/"; preceded by a <backslash>, "\/"; or within a bracket expression, "[/]". The start-condition '<' and '>' operators shall be special only in a start condition at the beginning of a regular expression; elsewhere in the regular expression they shall be treated as ordinary characters.

Actions in *lex*

The action to be taken when an ERE is matched can be a C program fragment or the special actions described below; the program fragment can contain one or more C statements, and can also include special actions. The empty C statement ';' shall be a valid action; any string in the *lex.yy.c* input that matches the pattern portion of such a rule is effectively ignored or skipped. However, the absence of an action shall not be valid, and the action *lex* takes in such a condition is undefined.

The specification for an action, including C statements and special actions, can extend across several lines if enclosed in braces:

```
ERE <one or more blanks> { program statement
                           program statement }
```

95408 The program statements shall not contain unbalanced curly brace preprocessing tokens.

95409 The default action when a string in the input to a **lex.yy.c** program is not matched by any
 95410 expression shall be to copy the string to the output. Because the default behavior of a program
 95411 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that
 95412 has just "%%" shall generate a C program that simply copies the input to the output unchanged.

95413 Four special actions shall be available:

95414 | ECHO; REJECT; BEGIN

95415 | The action '|' means that the action for the next rule is the action for this rule.
 95416 Unlike the other three actions, '|' cannot be enclosed in braces or be
 95417 <semicolon>-terminated; the application shall ensure that it is specified alone, with
 95418 no other actions.

95419 **ECHO;** Write the contents of the string *ytext* on the output.

95420 **REJECT;** Usually only a single expression is matched by a given string in the input.
 95421 **REJECT** means "continue to the next expression that matches the current input",
 95422 and shall cause whatever rule was the second choice after the current rule to be
 95423 executed for the same input. Thus, multiple rules can be matched and executed for
 95424 one input string or overlapping input strings. For example, given the regular
 95425 expressions "xyz" and "xy" and the input "xyz", usually only the regular
 95426 expression "xyz" would match. The next attempted match would start after **z**. If
 95427 the last action in the "xyz" rule is **REJECT**, both this rule and the "xy" rule
 95428 would be executed. The **REJECT** action may be implemented in such a fashion that
 95429 flow of control does not continue after it, as if it were equivalent to a **goto**
 95430 to another part of *yylex()*. The use of **REJECT** may result in somewhat larger and
 95431 slower scanners.

95432 **BEGIN** The action:

95433 BEGIN *newstate*;

95434 switches the state (start condition) to *newstate*. If the string *newstate* has not been
 95435 declared previously as a start condition in the *Definitions* section, the results are
 95436 unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

95437 The functions or macros described below are accessible to user code included in the *lex* input. It
 95438 is unspecified whether they appear in the C code output of *lex*, or are accessible only through the
 95439 **-ll** operand to *c99* (the *lex* library).

95440 **int yylex(void)**
 95441 Performs lexical analysis on the input; this is the primary function generated by the *lex*
 95442 utility. The function shall return zero when the end of input is reached; otherwise, it shall
 95443 return non-zero values (tokens) determined by the actions that are selected.

95444 **int yymore(void)**
 95445 When called, indicates that when the next input string is recognized, it is to be appended to
 95446 the current value of *ytext* rather than replacing it; the value in *yyleng* shall be adjusted
 95447 accordingly.

95448 **int yyles(int n)**
 95449 Retains *n* initial characters in *ytext*, NUL-terminated, and treats the remaining characters as
 95450 if they had not been read; the value in *yyleng* shall be adjusted accordingly.

95451 **int input(void)**
 95452 Returns the next character from the input, or zero on end-of-file. It shall obtain input from
 95453 the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning
 95454 has begun, the effect of altering the value of *yyin* is undefined. The character read shall be
 95455 removed from the input stream of the scanner without any processing by the scanner.

95456 **int unput(int c)**
 95457 Returns the character 'c' to the input; *yytext* and *yylen* are undefined until the next
 95458 expression is matched. The result of using *unput()* for more characters than have been input
 95459 is unspecified.

95460 The following functions shall appear only in the *lex* library accessible through the `-lI` operand;
 95461 they can therefore be redefined by a conforming application:

95462 **int yywrap(void)**
 95463 Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application
 95464 requires *yylex()* to continue processing with another source of input, then the application
 95465 can include a function *yywrap()*, which associates another file with the external variable
 95466 **FILE *yyin** and shall return a value of zero.

95467 **int main(int argc, char *argv[])**
 95468 Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to
 95469 perform application-specific operations, calling *yylex()* as applicable.

95470 Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin
 95471 with the prefix **yy** or **YY**.

95472 EXIT STATUS

95473 The following exit values shall be returned:

95474 0 Successful completion.

95475 >0 An error occurred.

95476 CONSEQUENCES OF ERRORS

95477 Default.

95478 APPLICATION USAGE

95479 Conforming applications are warned that in the *Rules* section, an ERE without an action is not
 95480 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or
 95481 runtime errors.

95482 The purpose of *input()* is to take characters off the input stream and discard them as far as the
 95483 lexical analysis is concerned. A common use is to discard the body of a comment once the
 95484 beginning of a comment is recognized.

95485 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*
 95486 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer
 95487 interpret the regular expressions given in the *lex* source according to the environment specified
 95488 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.
 95489 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the
 95490 lexical requirements of the input language being described, which is frequently locale-specific
 95491 anyway. (For example, writing an analyzer that is used for French text is not automatically
 95492 useful for processing other languages.)

95493 **EXAMPLES**

95494 The following is an example of a *lex* program that implements a rudimentary scanner for a
 95495 Pascal-like syntax:

```

95496     %{
95497     /* Need this for the call to atof() below. */
95498     #include <math.h>
95499     /* Need this for printf(), fopen(), and stdin below. */
95500     #include <stdio.h>
95501     %}

95502     DIGIT    [0-9]
95503     ID       [a-z][a-z0-9]*

95504     %%

95505     {DIGIT}+ {
95506         printf("An integer: %s (%d)\n", yytext,
95507             atoi(yytext));
95508     }

95509     {DIGIT}+"."{DIGIT}* {
95510         printf("A float: %s (%g)\n", yytext,
95511             atof(yytext));
95512     }

95513     if|then|begin|end|procedure|function {
95514         printf("A keyword: %s\n", yytext);
95515     }

95516     {ID}    printf("An identifier: %s\n", yytext);

95517     "+"|"-"|"*"|"/"    printf("An operator: %s\n", yytext);

95518     "{[^]\n}*" /* Eat up one-line comments. */

95519     [ \t\n]+ /* Eat up white space. */

95520     . printf("Unrecognized character: %s\n", yytext);

95521     %%

95522     int main(int argc, char *argv[])
95523     {
95524         ++argv, --argc; /* Skip over program name. */
95525         if (argc > 0)
95526             yyin = fopen(argv[0], "r");
95527         else
95528             yyin = stdin;

95529         yylex();
95530     }
  
```

95531 **RATIONALE**

95532 Even though the `-c` option and references to the C language are retained in this description, *lex*
 95533 may be generalized to other languages, as was done at one time for EFL, the Extended
 95534 FORTRAN Language. Since the *lex* input specification is essentially language-independent,
 95535 versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are
 95536 known historical implementations that do so.

95537 The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex*
95538 source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is
95539 assumed to be presented in the POSIX locale, but input and output are in the locale specified by
95540 the environment variables), then the tables in the lexical analyzer produced by *lex* would
95541 interpret EREs specified in the *lex* source in terms of the environment variables specified when
95542 *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs
95543 given in the *lex* source according to the environment specified when the lexical analyzer is
95544 executed, but this is not possible with the current *lex* technology.

95545 The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard
95546 use of escape sequences.

95547 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
95548 but this has been modified in this version.

95549 There is no detailed output format specification. The observed behavior of *lex* under four
95550 different historical implementations was that none of these implementations consistently
95551 reported the line numbers for error and warning messages. Furthermore, there was a desire that
95552 *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified
95553 avoids these formatting questions and problems with internationalization.

95554 Although the `%x` specifier for *exclusive* start conditions is not historical practice, it is believed to
95555 be a minor change to historical implementations and greatly enhances the usability of *lex*
95556 programs since it permits an application to obtain the expected functionality with fewer
95557 statements.

95558 The `%array` and `%pointer` declarations were added as a compromise between historical systems.
95559 The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in
95560 BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements
95561 are available for some scanners. Most historical programs should require no change in porting
95562 from one system to another because the string being referenced is null-terminated in both cases.
95563 (The method used by *flex* in its case is to null-terminate the token in place by remembering the
95564 character that used to come right after the token and replacing it before continuing on to the next
95565 scan.) Multi-file programs with external references to *yytext* outside the scanner source file
95566 should continue to operate on their historical systems, but would require one of the new
95567 declarations to be considered strictly portable.

95568 The description of EREs avoids unnecessary duplication of ERE details because their meanings
95569 within a *lex* ERE are the same as that for the ERE in this volume of POSIX.1-2017.

95570 The reason for the undefined condition associated with text beginning with a <blank> or within
95571 "`%{`" and "`%}`" delimiter lines appearing in the *Rules* section is historical practice. Both the BSD
95572 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the
95573 beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break*
95574 statement). In some cases, the System V *lex* generates an error message or a syntax error,
95575 depending on the form of indented input.

95576 The intention in breaking the list of functions into those that may appear in `lex.yy.c` versus those
95577 that only appear in `libl.a` is that only those functions in `libl.a` can be reliably redefined by a
95578 conforming application.

95579 The descriptions of standard output and standard error are somewhat complicated because
95580 historical *lex* implementations chose to issue diagnostic messages to standard output (unless `-t`
95581 was given). POSIX.1-2017 allows this behavior, but leaves an opening for the more expected
95582 behavior of using standard error for diagnostics. Also, the System V behavior of writing the
95583 statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The

- 95584 programmer can always precisely obtain the desired results by using either the `-t` or `-n` options.
- 95585 The OPERANDS section does not mention the use of `-` as a synonym for standard input; not all
95586 historical implementations support such usage for any of the *file* operands.
- 95587 A description of the *translation table* was deleted from early proposals because of its relatively
95588 low usage in historical applications.
- 95589 The change to the definition of the *input()* function that allows buffering of input presents the
95590 opportunity for major performance gains in some applications.
- 95591 The following examples clarify the differences between *lex* regular expressions and regular
95592 expressions appearing elsewhere in this volume of POSIX.1-2017. For regular expressions of the
95593 form "*r/x*", the string matching *r* is always returned; confusion may arise when the beginning
95594 of *x* matches the trailing portion of *r*. For example, given the regular expression "*a*b/cc*" and
95595 the input "*aaabcc*", *yytext* would contain the string "*aaab*" on this match. But given the
95596 regular expression "*x*/xy*" and the input "*xxxxy*", the token *xxx*, not *xx*, is returned by some
95597 implementations because *xxx* matches "*x**".
- 95598 In the rule "*ab*/bc*", the "*b**" at the end of *r* extends *r*'s match into the beginning of the
95599 trailing context, so the result is unspecified. If this rule were "*ab/bc*", however, the rule
95600 matches the text "*ab*" when it is followed by the text "*bc*". In this latter case, the matching of *r*
95601 cannot extend into the beginning of *x*, so the result is specified.
- 95602 **FUTURE DIRECTIONS**
- 95603 None.
- 95604 **SEE ALSO**
- 95605 [c99, ed, yacc](#)
- 95606 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Chapter 9](#) (on page 181), [Section 12.2](#) (on
95607 page 216)
- 95608 **CHANGE HISTORY**
- 95609 First released in Issue 2.
- 95610 **Issue 6**
- 95611 This utility is marked as part of the C-Language Development Utilities option.
- 95612 The obsolescent `-c` option is removed.
- 95613 The normative text is reworded to avoid use of the term “must” for application requirements.
- 95614 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/14 is applied, removing text describing
95615 behavior on systems with bytes consisting of more than eight bits.
- 95616 **Issue 7**
- 95617 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for
95618 generated code to conform to the ISO C standard.
- 95619 Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language
95620 trigraphs and curly brace preprocessing tokens.
- 95621 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
95622 apply.
- 95623 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

95624 **NAME**

95625 link ‡callink() function

95626 **SYNOPSIS**95627 XSI link *file1 file2*95628 **DESCRIPTION**95629 The *link* utility shall perform the function call:95630 link(*file1*, *file2*);95631 A user may need appropriate privileges to invoke the *link* utility.95632 **OPTIONS**

95633 None.

95634 **OPERANDS**

95635 The following operands shall be supported:

95636 *file1* The pathname of an existing file.95637 *file2* The pathname of the new directory entry to be created.95638 **STDIN**

95639 Not used.

95640 **INPUT FILES**

95641 Not used.

95642 **ENVIRONMENT VARIABLES**95643 The following environment variables shall affect the execution of *link*:95644 *LANG* Provide a default value for the internationalization variables that are unset or null.
95645 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
95646 variables used to determine the values of locale categories.)95647 *LC_ALL* If set to a non-empty string value, override the values of all the other
95648 internationalization variables.95649 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
95650 characters (for example, single-byte as opposed to multi-byte characters in
95651 arguments).95652 *LC_MESSAGES*95653 Determine the locale that should be used to affect the format and contents of
95654 diagnostic messages written to standard error.95655 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.95656 **ASYNCHRONOUS EVENTS**

95657 Default.

95658 **STDOUT**

95659 None.

95660 **STDERR**

95661 The standard error shall be used only for diagnostic messages.

95662 **OUTPUT FILES**

95663 None.

95664 **EXTENDED DESCRIPTION**

95665 None.

95666 **EXIT STATUS**

95667 The following exit values shall be returned:

95668 0 Successful completion.

95669 >0 An error occurred.

95670 **CONSEQUENCES OF ERRORS**

95671 Default.

95672 **APPLICATION USAGE**

95673 None.

95674 **EXAMPLES**

95675 None.

95676 **RATIONALE**

95677 None.

95678 **FUTURE DIRECTIONS**

95679 None.

95680 **SEE ALSO**95681 *ln, unlink*95682 XBD [Chapter 8](#) (on page 173)95683 XSH *link()*95684 **CHANGE HISTORY**

95685 First released in Issue 5.

95686 **NAME**

95687 ln ‡link files

95688 **SYNOPSIS**95689 ln [-fs] [-L|-P] *source_file target_file*95690 ln [-fs] [-L|-P] *source_file... target_dir*95691 **DESCRIPTION**

95692 In the first synopsis form, the *ln* utility shall create a new directory entry (link) at the destination
 95693 path specified by the *target_file* operand. If the *-s* option is specified, a symbolic link shall be
 95694 created for the file specified by the *source_file* operand. This first synopsis form shall be assumed
 95695 when the final operand does not name an existing directory; if more than two operands are
 95696 specified and the final is not an existing directory, an error shall result.

95697 In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the *-s*
 95698 option is specified a symbolic link, for each file specified by a *source_file* operand, at a
 95699 destination path in the existing directory named by *target_dir*.

95700 If the last operand specifies an existing file of a type not specified by the System Interfaces
 95701 volume of POSIX.1-2017, the behavior is implementation-defined.

95702 The corresponding destination path for each *source_file* shall be the concatenation of the target
 95703 directory pathname, a <slash> character if the target directory pathname did not end in a
 95704 <slash>, and the last pathname component of the *source_file*. The second synopsis form shall be
 95705 assumed when the final operand names an existing directory.

95706 For each *source_file*:

- 95707 1. If the destination path exists and was created by a previous step, it is unspecified whether
 95708 *ln* shall write a diagnostic message to standard error, do nothing more with the current
 95709 *source_file*, and go on to any remaining *source_files*; or will continue processing the current
 95710 *source_file*. If the destination path exists:
 - 95711 a. If the *-f* option is not specified, *ln* shall write a diagnostic message to standard
 95712 error, do nothing more with the current *source_file*, and go on to any remaining
 95713 *source_files*.
 - 95714 b. If the destination path names the same directory entry as the current *source_file* *ln*
 95715 shall write a diagnostic message to standard error, do nothing more with the
 95716 current *source_file*, and go on to any remaining *source_files*.
 - 95717 c. Actions shall be performed equivalent to the *unlink()* function defined in the
 95718 System Interfaces volume of POSIX.1-2017, called using the destination path as the
 95719 *path* argument. If this fails for any reason, *ln* shall write a diagnostic message to
 95720 standard error, do nothing more with the current *source_file*, and go on to any
 95721 remaining *source_files*.
- 95722 2. If the *-s* option is specified, actions shall be performed equivalent to the *symlink()*
 95723 function with *source_file* as the *path1* argument and the destination path as the *path2*
 95724 argument. The *ln* utility shall do nothing more with *source_file* and shall go on to any
 95725 remaining files.
- 95726 3. If *source_file* is a symbolic link:
 - 95727 a. If the *-P* option is in effect, actions shall be performed equivalent to the *linkat()*
 95728 function with *source_file* as the *path1* argument, the destination path as the *path2*
 95729 argument, *AT_FDCWD* as the *fd1* and *fd2* arguments, and zero as the *flag*
 95730 argument.

- 95731 b. If the `-L` option is in effect, actions shall be performed equivalent to the `linkat()`
 95732 function with `source_file` as the `path1` argument, the destination path as the `path2`
 95733 argument, `AT_FDCWD` as the `fd1` and `fd2` arguments, and
 95734 `AT_SYMLINK_FOLLOW` as the `flag` argument.

95735 The `ln` utility shall do nothing more with `source_file` and shall go on to any remaining files.

- 95736 4. Actions shall be performed equivalent to the `link()` function defined in the System
 95737 Interfaces volume of POSIX.1-2017 using `source_file` as the `path1` argument, and the
 95738 destination path as the `path2` argument.

95739 **OPTIONS**

95740 The `ln` utility shall conform to XBD [Section 12.2](#) (on page 216).

95741 The following options shall be supported:

- 95742 **-f** Force existing destination pathnames to be removed to allow the link.
- 95743 **-L** For each `source_file` operand that names a file of type symbolic link, create a (hard)
 95744 link to the file referenced by the symbolic link.
- 95745 **-P** For each `source_file` operand that names a file of type symbolic link, create a (hard)
 95746 link to the symbolic link itself.
- 95747 **-s** Create symbolic links instead of hard links. If the `-s` option is specified, the `-L` and
 95748 **-P** options shall be silently ignored.

95749 Specifying more than one of the mutually-exclusive options `-L` and `-P` shall not be considered
 95750 an error. The last option specified shall determine the behavior of the utility (unless the `-s`
 95751 option causes it to be ignored).

95752 If the `-s` option is not specified and neither a `-L` nor a `-P` option is specified, it is
 95753 implementation-defined which of the `-L` and `-P` options will be used as the default.

95754 **OPERANDS**

95755 The following operands shall be supported:

- 95756 `source_file` A pathname of a file to be linked. If the `-s` option is specified, no restrictions on the
 95757 type of file or on its existence shall be made. If the `-s` option is not specified,
 95758 whether a directory can be linked is implementation-defined.
- 95759 `target_file` The pathname of the new directory entry to be created.
- 95760 `target_dir` A pathname of an existing directory in which the new directory entries are created.

95761 **STDIN**

95762 Not used.

95763 **INPUT FILES**

95764 None.

95765 **ENVIRONMENT VARIABLES**

95766 The following environment variables shall affect the execution of `ln`:

- 95767 **LANG** Provide a default value for the internationalization variables that are unset or null.
 95768 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 95769 variables used to determine the values of locale categories.)
- 95770 **LC_ALL** If set to a non-empty string value, override the values of all the other
 95771 internationalization variables.

95772 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 95773 characters (for example, single-byte as opposed to multi-byte characters in
 95774 arguments).

95775 *LC_MESSAGES*
 95776 Determine the locale that should be used to affect the format and contents of
 95777 diagnostic messages written to standard error.

95778 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

95779 **ASYNCHRONOUS EVENTS**
 95780 Default.

95781 **STDOUT**
 95782 Not used.

95783 **STDERR**
 95784 The standard error shall be used only for diagnostic messages.

95785 **OUTPUT FILES**
 95786 None.

95787 **EXTENDED DESCRIPTION**
 95788 None.

95789 **EXIT STATUS**
 95790 The following exit values shall be returned:
 95791 0 All the specified files were linked successfully.
 95792 >0 An error occurred.

95793 **CONSEQUENCES OF ERRORS**
 95794 Default.

95795 **APPLICATION USAGE**
 95796 None.

95797 **EXAMPLES**
 95798 None.

95799 **RATIONALE**
 95800 The CONSEQUENCES OF ERRORS section does not require *ln -f a b* to remove *b* if a
 95801 subsequent link operation would fail.

95802 Some historic versions of *ln* (including the one specified by the SVID) unlink the destination file,
 95803 if it exists, by default. If the mode does not permit writing, these versions prompt for
 95804 confirmation before attempting the unlink. In these versions the *-f* option causes *ln* not to
 95805 attempt to prompt for confirmation.

95806 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself
 95807 is not writable (although the directory must be). Early proposals specified this functionality.

95808 This volume of POSIX.1-2017 does not allow the *ln* utility to unlink existing destination paths by
 95809 default for the following reasons:

95810 The *ln* utility has historically been used to provide locking for shell applications, a usage
 95811 that is incompatible with *ln* unlinking the destination path by default. There was no
 95812 corresponding technical advantage to adding this functionality.

95813 This functionality gave *ln* the ability to destroy the link structure of files, which changes
95814 the historical behavior of *ln*.

95815 This functionality is easily replicated with a combination of *rm* and *ln*.

95816 It is not historical practice in many systems; BSD and BSD-derived systems do not support
95817 this behavior. Unfortunately, whichever behavior is selected can cause scripts written
95818 expecting the other behavior to fail.

95819 It is preferable that *ln* perform in the same manner as the *link()* function, which does not
95820 permit the target to exist already.

95821 This volume of POSIX.1-2017 retains the *-f* option to provide support for shell scripts depending
95822 on the SVID semantics. It seems likely that shell scripts would not be written to handle
95823 prompting by *ln* and would therefore have specified the *-f* option.

95824 The *-f* option is an undocumented feature of many historical versions of the *ln* utility, allowing
95825 linking to directories. These versions require modification.

95826 Early proposals of this volume of POSIX.1-2017 also required a *-i* option, which behaved like the
95827 *-i* options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not
95828 historical practice for the *ln* utility and has been omitted.

95829 The *-L* and *-P* options allow for implementing both common behaviors of the *ln* utility. Earlier
95830 versions of this standard did not specify these options and required the behavior now described
95831 for the *-L* option. Many systems by default or as an alternative provided a non-conforming *ln*
95832 utility with the behavior now described for the *-P* option. Since applications could not rely on *ln*
95833 following links in practice, the *-L* and *-P* options were added to specify the desired behavior for
95834 the application.

95835 The *-L* and *-P* options are ignored when *-s* is specified in order to allow an alias to be created to
95836 alter the default behavior when creating hard links (for example, *alias ln='ln -L'*). They serve no
95837 purpose when *-s* is specified, since *source_file* is then just a string to be used as the contents of
95838 the created symbolic link and need not exist as a file.

95839 The specification ensures that *ln a a* with or without the *-f* option will not unlink the file *a*.
95840 Earlier versions of this standard were unclear in this case.

95841 **FUTURE DIRECTIONS**

95842 None.

95843 **SEE ALSO**

95844 *chmod*, *find*, *pax*, *rm*

95845 XBD Chapter 8 (on page 173), Section 12.2 (on page 216)

95846 XSH *link()*, *unlink()*

95847 **CHANGE HISTORY**

95848 First released in Issue 2.

95849 **Issue 6**

95850 The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b
95851 draft standard.

95852 **Issue 7**

95853 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

95854 SD5-XCU-ERN-27 is applied, adding a new paragraph to the RATIONALE.

95855 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

95856 The **-L** and **-P** options are added to make it implementation-defined whether the *ln* utility
95857 follows symbolic links.

95858 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0096 [136] is applied.

95859 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0113 [930] is applied.

95860 **NAME**

95861 locale ‡get locale-specific information

95862 **SYNOPSIS**

95863 locale [-a|-m]

95864 locale [-ck] name...

95865 **DESCRIPTION**

95866 The *locale* utility shall write information about the current locale environment, or all public
 95867 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by
 95868 the implementation that is accessible to the application.

95869 When *locale* is invoked without any arguments, it shall summarize the current locale
 95870 environment for each locale category as determined by the settings of the environment variables
 95871 defined in XBD [Chapter 7](#) (on page 135).

95872 When invoked with operands, it shall write values that have been assigned to the keywords in
 95873 the locale categories, as follows:

95874 Specifying a keyword name shall select the named keyword and the category containing
 95875 that keyword.

95876 Specifying a category name shall select the named category and all keywords in that
 95877 category.

95878 **OPTIONS**95879 The *locale* utility shall conform to XBD [Section 12.2](#) (on page 216).

95880 The following options shall be supported:

95881 **-a** Write information about all available public locales. The available locales shall
 95882 include **POSIX**, representing the POSIX locale. The manner in which the
 95883 implementation determines what other locales are available is implementation-
 95884 defined.

95885 **-c** Write the names of selected locale categories; see the **STDOUT** section. The **-c**
 95886 option increases readability when more than one category is selected (for example,
 95887 via more than one keyword name or via a category name). It is valid both with
 95888 and without the **-k** option.

95889 **-k** Write the names and values of selected keywords. The implementation may omit
 95890 values for some keywords; see the **OPERANDS** section.

95891 **-m** Write names of available charmaps; see XBD [Section 6.1](#) (on page 125).

95892 **OPERANDS**

95893 The following operand shall be supported:

95894 *name* The name of a locale category as defined in XBD [Chapter 7](#) (on page 135), the name
 95895 of a keyword in a locale category, or the reserved name **charmap**. The named
 95896 category or keyword shall be selected for output. If a single *name* represents both a
 95897 locale category name and a keyword name in the current locale, the results are
 95898 unspecified. Otherwise, both category and keyword names can be specified as
 95899 *name* operands, in any sequence. It is implementation-defined whether any
 95900 keyword values are written for the categories *LC_CTYPE* and *LC_COLLATE*.

95901 **STDIN**

95902 Not used.

95903 **INPUT FILES**

95904 None.

95905 **ENVIRONMENT VARIABLES**95906 The following environment variables shall affect the execution of *locale*:

95907 *LANG* Provide a default value for the internationalization variables that are unset or null.
 95908 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 95909 variables used to determine the values of locale categories.)

95910 *LC_ALL* If set to a non-empty string value, override the values of all the other
 95911 internationalization variables.

95912 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 95913 characters (for example, single-byte as opposed to multi-byte characters in
 95914 arguments and input files).

95915 *LC_MESSAGES*

95916 Determine the locale that should be used to affect the format and contents of
 95917 diagnostic messages written to standard error.

95918 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

95919 XSI The application shall ensure that the *LANG*, *LC_**, and *NLSPATH* environment variables specify
 95920 the current locale environment to be written out; they shall be used if the **-a** option is not
 95921 specified.

95922 **ASYNCHRONOUS EVENTS**

95923 Default.

95924 **STDOUT**95925 The *LANG* variable shall be written first using the format:95926 "LANG=%s\n", *<value>*95927 If *LANG* is not set or is an empty string, the value is the empty string.

95928 If *locale* is invoked without any options or operands, the names and values of the *LC_**
 95929 environment variables described in this volume of POSIX.1-2017 shall be written to the standard
 95930 output, one variable per line, and each line using the following format. Only those variables set
 95931 in the environment and not overridden by *LC_ALL* shall be written using this format:

95932 "%s=%s\n", *<variable_name>*, *<value>*

95933 The names of those *LC_** variables associated with locale categories defined in this volume of
 95934 POSIX.1-2017 that are not set in the environment or are overridden by *LC_ALL* shall be written
 95935 in the following format:

95936 "%s=\"%s\"\n", *<variable_name>*, *<implied value>*

95937 The *<implied value>* shall be the name of the locale that has been selected for that category by the
 95938 implementation, based on the values in *LANG* and *LC_ALL*, as described in XBD [Chapter 8](#) (on
 95939 page 173).

95940 The *<value>* and *<implied value>* shown above shall be properly quoted for possible later reentry
 95941 to the shell. The *<value>* shall not be quoted using double-quotes (so that it can be distinguished
 95942 by the user from the *<implied value>* case, which always requires double-quotes).

95943 The `LC_ALL` variable shall be written last, using the first format shown above. If it is not set, it
 95944 shall be written as:

95945 `"LC_ALL=\n"`

95946 If any arguments are specified:

95947 1. If the `-a` option is specified, the names of all the public locales shall be written, each in the
 95948 following format:

95949 `"%s\n", <locale name>`

95950 2. If the `-c` option is specified, the names of all selected categories shall be written, each in
 95951 the following format:

95952 `"%s\n", <category name>`

95953 If keywords are also selected for writing (see following items), the category name output
 95954 shall precede the keyword output for that category.

95955 If the `-c` option is not specified, the names of the categories shall not be written; only the
 95956 keywords, as selected by the `<name>` operand, shall be written.

95957 3. If the `-k` option is specified, the names and values of selected keywords shall be written.
 95958 If a value is non-numeric and is not a compound keyword value, it shall be written in the
 95959 following format:

95960 `"%s=\"%s\"\n", <keyword name>, <keyword value>`

95961 If a value is a non-numeric compound keyword value, it shall either be written in the
 95962 format:

95963 `"%s=\"%s\"\n", <keyword name>, <keyword value>`

95964 where the `<keyword value>` is a single string of values separated by `<semicolon>`
 95965 characters, or it shall be written in the format:

95966 `"%s=%s\n", <keyword name>, <keyword value>`

95967 where the `<keyword value>` is encoded as a set of strings, each enclosed in double-
 95968 quotation-marks, separated by `<semicolon>` characters.

95969 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
 95970 `localedef -f` option when the locale was created shall be written, with the word **charmap** as
 95971 `<keyword name>`.

95972 If a value is numeric, it shall be written in one of the following formats:

95973 `"%s=%d\n", <keyword name>, <keyword value>`

95974 `"%s=%c%o\n", <keyword name>, <escape character>, <keyword value>`

95975 `"%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>`

95976 where the `<escape character>` is that identified by the **escape_char** keyword in the current
 95977 locale; see XBD [Section 7.3](#) (on page 136).

95978 Compound keyword values (list entries) shall be separated in the output by `<semicolon>`
 95979 characters. When included in keyword values, the `<semicolon>`, `<backslash>`, double-
 95980 quote, and any control character shall be preceded (escaped) with the escape character.

95981 4. If the `-k` option is not specified, selected keyword values shall be written, each in the
 95982 following format:

95983 `"%s\n", <keyword value>`

95984 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
 95985 `localedef -f` option when the locale was created shall be written.

95986 5. If the `-m` option is specified, then a list of all available charmaps shall be written, each in
 95987 the format:

95988 `"%s\n", <charmap>`

95989 where `<charmap>` is in a format suitable for use as the option-argument to the `localedef -f`
 95990 option.

95991 **STDERR**

95992 The standard error shall be used only for diagnostic messages.

95993 **OUTPUT FILES**

95994 None.

95995 **EXTENDED DESCRIPTION**

95996 None.

95997 **EXIT STATUS**

95998 The following exit values shall be returned:

95999 0 All the requested information was found and output successfully.

96000 >0 An error occurred.

96001 **CONSEQUENCES OF ERRORS**

96002 Default.

96003 **APPLICATION USAGE**

96004 If the `LANG` environment variable is not set or set to an empty value, or one of the `LC_*`
 96005 environment variables is set to an unrecognized value, the actual locales assumed (if any) are
 96006 implementation-defined as described in XBD [Chapter 8](#) (on page 173).

96007 Implementations are not required to write out the actual values for keywords in the categories
 96008 `LC_CTYPE` and `LC_COLLATE`; however, they must write out the categories (allowing an
 96009 application to determine, for example, which character classes are available).

96010 **EXAMPLES**

96011 In the following examples, the assumption is that locale environment variables are set as
 96012 follows:

96013 `LANG=locale_x`

96014 `LC_COLLATE=locale_y`

96015 The command `locale` would result in the following output:

96016 `LANG=locale_x`

96017 `LC_CTYPE="locale_x"`

96018 `LC_COLLATE=locale_y`

96019 `LC_TIME="locale_x"`

96020 `LC_NUMERIC="locale_x"`

96021 `LC_MONETARY="locale_x"`

96022 `LC_MESSAGES="locale_x"`

96023 `LC_ALL=`

96024 The order of presentation of the categories is not specified by this volume of POSIX.1-2017.

96025 The command:

```
96026 LC_ALL=POSIX locale -ck decimal_point
```

96027 would produce:

```
96028 LC_NUMERIC
96029 decimal_point="."
```

96030 The following command shows an application of *locale* to determine whether a user-supplied
96031 response is affirmative:

```
96032 printf 'Prompt for response: '
96033 read response
96034 if printf "%s\n" "$response" | grep -- -Eq "$(locale yesexpr)"
96035 then
96036     affirmative processing goes here
96037 else
96038     non-affirmative processing goes here
96039 fi
```

96040 RATIONALE

96041 The output for categories *LC_CTYPE* and *LC_COLLATE* has been made implementation-defined
96042 because there is a questionable value in having a shell script receive an entire array of characters.
96043 It is also difficult to return a logical collation description, short of returning a complete *localedef*
96044 source.

96045 The **-m** option was included to allow applications to query for the existence of charmaps. The
96046 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the
96047 system.

96048 The **-c** option was included for readability when more than one category is selected (for
96049 example, via more than one keyword name or via a category name). It is valid both with and
96050 without the **-k** option.

96051 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the
96052 current locale was created, was included to allow applications needing the information to
96053 retrieve it.

96054 According to XBD [Section 6.1](#) (on page 125), the standard requires that all supported locales
96055 must have the same encoding for <period> and <slash>, because these two characters are used
96056 within the locale-independent pathname resolution sequence. Therefore, it would be an error if
96057 *locale -a* listed both ASCII and EBCDIC-based locales, since those two encodings do not share
96058 the same representation for either <period> or <slash>. Any system that supports both
96059 environments would be expected to provide two POSIX locales, one in either codeset, where
96060 only the locales appropriate to the current environment can be visible at a time. In an XSI-
96061 compliant implementation, the *dd* utility is the only portable means for performing conversions
96062 between the two character sets.

96063 FUTURE DIRECTIONS

96064 None.

96065 SEE ALSO

96066 [localedef](#)

96067 XBD [Section 6.1](#) (on page 125), [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [Section 12.2](#) (on
96068 page 216)

96069 **CHANGE HISTORY**

96070 First released in Issue 4.

96071 **Issue 5**

96072 The FUTURE DIRECTIONS section is added.

96073 **Issue 6**

96074 The normative text is reworded to avoid use of the term “must” for application requirements.

96075 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error
96076 in the STDOUT section.

96077 **Issue 7**

96078 Austin Group Interpretations 1003.1-2001 #017, #021, and #088 are applied, clarifying the
96079 standard output for the `-k` option when `LANG` is not set or is an empty string.

96080 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

96081 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0097 [291] is applied.

96082 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0114 [941] is applied.

96083 **NAME**

96084 localedef — define locale environment

96085 **SYNOPSIS**96086 localedef [-c] [-f *charmap*] [-i *sourcefile*] [-u *code_set_name*] *name*96087 **DESCRIPTION**

96088 The *localedef* utility shall convert source definitions for locale categories into a format usable by
 96089 the functions and utilities whose operational behavior is determined by the setting of the locale
 96090 environment variables defined in XBD [Chapter 7](#) (on page 135). It is implementation-defined
 96091 whether users have the capability to create new locales, in addition to those supplied by the
 96092 implementation. If the symbolic constant POSIX2_LOCALEDEF is defined, the system supports
 96093 XSI the creation of new locales. On XSI-conformant systems, the symbolic constant
 96094 POSIX2_LOCALEDEF shall be defined.

96095 The utility shall read source definitions for one or more locale categories belonging to the same
 96096 locale from the file named in the *-i* option (if specified) or from standard input.

96097 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or
 96098 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may
 96099 restrict the capability to create or modify public locales to users with appropriate privileges.

96100 Each category source definition shall be identified by the corresponding environment variable
 96101 name and terminated by an **END** *category-name* statement. The following categories shall be
 96102 supported. In addition, the input may contain source for implementation-defined categories.

96103 *LC_CTYPE* Defines character classification and case conversion.

96104 *LC_COLLATE*
 96105 Defines collation rules.

96106 *LC_MONETARY*
 96107 Defines the format and symbols used in formatting of monetary information.

96108 *LC_NUMERIC*
 96109 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary
 96110 numeric editing.

96111 *LC_TIME* Defines the format and content of date and time information.

96112 *LC_MESSAGES*
 96113 Defines the format and values of affirmative and negative responses.

96114 **OPTIONS**

96115 The *localedef* utility shall conform to XBD [Section 12.2](#) (on page 216).

96116 The following options shall be supported:

96117 **-c** Create permanent output even if warning messages have been issued.

96118 **-f** *charmap* Specify the pathname of a file containing a mapping of character symbols and
 96119 collating element symbols to actual character encodings. The format of the
 96120 *charmap* is described in XBD [Section 6.4](#) (on page 129). The application shall ensure
 96121 that this option is specified if symbolic names (other than collating symbols
 96122 defined in a **collating-symbol** keyword) are used. If the *-f* option is not present, an
 96123 implementation-defined character mapping shall be used.

96124 **-i** *inputfile* The pathname of a file containing the source definitions. If this option is not
 96125 present, source definitions shall be read from standard input. The format of the
 96126 *inputfile* is described in XBD [Section 7.3](#) (on page 136).

96127 **-u** *code_set_name*
 96128 Specify the name of a codeset used as the target mapping of character symbols and
 96129 collating element symbols whose encoding values are defined in terms of the
 96130 ISO/IEC 10646-1:2000 standard position constant values.

96131 OPERANDS

96132 The following operand shall be supported:

96133 *name* Identifies the locale; see XBD [Chapter 7](#) (on page 135) for a description of the use of
 96134 this name. If the name contains one or more <slash> characters, *name* shall be
 96135 interpreted as a pathname where the created locale definitions shall be stored. If
 96136 *name* does not contain any <slash> characters, the interpretation of the name is
 96137 implementation-defined and the locale shall be public. The ability to create public
 96138 locales in this way may be restricted to users with appropriate privileges. (As a
 96139 consequence of specifying one *name*, although several categories can be processed
 96140 in one execution, only categories belonging to the same locale can be processed.)

96141 STDIN

96142 Unless the **-i** option is specified, the standard input shall be a text file containing one or more
 96143 locale category source definitions, as described in XBD [Section 7.3](#) (on page 136). When lines are
 96144 continued using the escape character mechanism, there is no limit to the length of the
 96145 accumulated continued line.

96146 INPUT FILES

96147 The character set mapping file specified as the *charmap* option-argument is described in XBD
 96148 [Section 6.4](#) (on page 129). If a locale category source definition contains a **copy** statement, as
 96149 defined in XBD [Chapter 7](#) (on page 135), and the **copy** statement names a valid, existing locale,
 96150 then *localedef* shall behave as if the source definition had contained a valid category source
 96151 definition for the named locale.

96152 ENVIRONMENT VARIABLES

96153 The following environment variables shall affect the execution of *localedef*:

96154 **LANG** Provide a default value for the internationalization variables that are unset or null.
 96155 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 96156 variables used to determine the values of locale categories.)

96157 **LC_ALL** If set to a non-empty string value, override the values of all the other
 96158 internationalization variables.

96159 **LC_COLLATE**
 96160 (This variable has no affect on *localedef*; the POSIX locale is used for this category.)

96161 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 96162 characters (for example, single-byte as opposed to multi-byte characters in
 96163 arguments and input files). This variable has no affect on the processing of *localedef*
 96164 input data; the POSIX locale is used for this purpose, regardless of the value of this
 96165 variable.

96166 **LC_MESSAGES**
 96167 Determine the locale that should be used to affect the format and contents of
 96168 diagnostic messages written to standard error.

96169 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

96170 **ASYNCHRONOUS EVENTS**

96171 Default.

96172 **STDOUT**

96173 The utility shall report all categories successfully processed, in an unspecified format.

96174 **STDERR**

96175 The standard error shall be used only for diagnostic messages.

96176 **OUTPUT FILES**96177 The format of the created output is unspecified. If the *name* operand does not contain a <slash>, 96178 the existence of an output file for the locale is unspecified.96179 **EXTENDED DESCRIPTION**96180 When the **-u** option is used, the *code_set_name* option-argument shall be interpreted as an 96181 implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard 96182 position constant values shall be converted via an implementation-defined method. Both the 96183 ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal, 96184 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset 96185 represented by the implementation-defined name can be any codeset that is supported by the 96186 implementation.96187 When conflicts occur between the *charmap* specification of <*code_set_name*>, <*mb_cur_max*>, or 96188 <*mb_cur_min*> and the implementation-defined interpretation of these respective items for the 96189 codeset represented by the **-u** option-argument *code_set_name*, the result is unspecified.96190 When conflicts occur between the *charmap* encoding values specified for symbolic names of 96191 characters of the portable character set and the implementation-defined assignment of character 96192 encoding values, the result is unspecified.96193 If a non-printable character in the *charmap* has a width specified that is not **-1**, the result will be 96194 undefined.96195 **EXIT STATUS**

96196 The following exit values shall be returned:

- 96197 0 No errors occurred and the locales were successfully created.
- 96198 1 Warnings occurred and the locales were successfully created.
- 96199 2 The locale specification exceeded implementation limits or the coded character set or sets 96200 used were not supported by the implementation, and no locale was created.
- 96201 3 The capability to create new locales is not supported by the implementation.
- 96202 >3 Warnings or errors occurred and no output was created.

96203 **CONSEQUENCES OF ERRORS**

96204 If an error is detected, no permanent output shall be created.

96205 If warnings occur, permanent output shall be created if the **-c** option was specified. The 96206 following conditions shall cause warning messages to be issued:96207 If a symbolic name not found in the *charmap* file is used for the descriptions of the 96208 *LC_CTYPE* or *LC_COLLATE* categories (for other categories, this shall be an error 96209 condition).96210 If the number of operands to the **order** keyword exceeds the {*COLL_WEIGHTS_MAX*} 96211 limit.

96212 If optional keywords not supported by the implementation are present in the source.

96213 Other implementation-defined conditions may also cause warnings.

96214 APPLICATION USAGE

96215 The *charmap* definition is optional, and is contained outside the locale definition. This allows
96216 both completely self-defined source files, and generic sources (applicable to more than one
96217 codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the
96218 portable character set. As explained in XBD [Section 6.4](#) (on page 129), it is implementation-
96219 defined whether or not users or applications can provide additional character set description
96220 files. Therefore, the `-f` option might be operable only when an implementation-defined *charmap*
96221 is named.

96222 EXAMPLES

96223 None.

96224 RATIONALE

96225 The output produced by the *localedef* utility is implementation-defined. The *name* operand is
96226 used to identify the specific locale. (As a consequence, although several categories can be
96227 processed in one execution, only categories belonging to the same locale can be processed.)

96228 FUTURE DIRECTIONS

96229 None.

96230 SEE ALSO

96231 [locale](#)

96232 XBD [Section 6.4](#) (on page 129), [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [Section 12.2](#) (on
96233 page 216)

96234 CHANGE HISTORY

96235 First released in Issue 4.

96236 Issue 6

96237 The `-u` option is added, as specified in the IEEE P1003.2b draft standard.

96238 The normative text is reworded to avoid use of the term “must” for application requirements.

96239 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/15 is applied, rewording text in the
96240 OPERANDS section describing the ability to create public locales.

96241 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/16 is applied, making the text consistent
96242 with the descriptions of **WIDTH** and **WIDTH_DEFAULT** in the Base Definitions volume of
96243 POSIX.1-2017.

96244 Issue 7

96245 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

96246 **NAME**96247 `logger` ‡log messages96248 **SYNOPSIS**96249 `logger string...`96250 **DESCRIPTION**

96251 The *logger* utility saves a message, in an unspecified manner and format, containing the *string*
 96252 operands provided by the user. The messages are expected to be evaluated later by personnel
 96253 performing system administration tasks.

96254 It is implementation-defined whether messages written in locales other than the POSIX locale
 96255 are effective.

96256 **OPTIONS**

96257 None.

96258 **OPERANDS**

96259 The following operand shall be supported:

96260 *string* One of the string arguments whose contents are concatenated together, in the order
 96261 specified, separated by single <space> characters.

96262 **STDIN**

96263 Not used.

96264 **INPUT FILES**

96265 None.

96266 **ENVIRONMENT VARIABLES**96267 The following environment variables shall affect the execution of *logger*:

96268 *LANG* Provide a default value for the internationalization variables that are unset or null.
 96269 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 96270 variables used to determine the values of locale categories.)

96271 *LC_ALL* If set to a non-empty string value, override the values of all the other
 96272 internationalization variables.

96273 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 96274 characters (for example, single-byte as opposed to multi-byte characters in
 96275 arguments).

96276 *LC_MESSAGES*

96277 Determine the locale that should be used to affect the format and contents of
 96278 diagnostic messages written to standard error. (This means diagnostics from *logger*
 96279 to the user or application, not diagnostic messages that the user is sending to the
 96280 system administrator.)

96281 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

96282 **ASYNCHRONOUS EVENTS**

96283 Default.

96284 **STDOUT**

96285 Not used.

96286 STDERR

96287 The standard error shall be used only for diagnostic messages.

96288 OUTPUT FILES

96289 Unspecified.

96290 EXTENDED DESCRIPTION

96291 None.

96292 EXIT STATUS

96293 The following exit values shall be returned:

96294 0 Successful completion.

96295 >0 An error occurred.

96296 CONSEQUENCES OF ERRORS

96297 Default.

96298 APPLICATION USAGE

96299 This utility allows logging of information for later use by a system administrator or programmer
96300 in determining why non-interactive utilities have failed. The locations of the saved messages,
96301 their format, and retention period are all unspecified. There is no method for a conforming
96302 application to read messages, once written.

96303 EXAMPLES

96304 A batch application, running non-interactively, tries to read a configuration file and fails; it may
96305 attempt to notify the system administrator with:

96306 `logger myname: unable to read file foo. [timestamp]`

96307 RATIONALE

96308 The standard developers believed strongly that some method of alerting administrators to errors
96309 was necessary. The obvious example is a batch utility, running non-interactively, that is unable to
96310 read its configuration files or that is unable to create or write its results file. However, the
96311 standard developers did not wish to define the format or delivery mechanisms as they have
96312 historically been (and will probably continue to be) very system-specific, as well as involving
96313 functionality clearly outside the scope of this volume of POSIX.1-2017.

96314 The text with *LC_MESSAGES* about diagnostic messages means diagnostics from *logger* to the
96315 user or application, not diagnostic messages that the user is sending to the system administrator.

96316 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

96317 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient
96318 justification to exclude these utilities from this volume of POSIX.1-2017. It is also arguable that
96319 they are, in fact, testable, but that the tests themselves are not portable.

96320 FUTURE DIRECTIONS

96321 None.

96322 SEE ALSO

96323 *lp*, *mailx*, *write*

96324 XBD [Chapter 8](#) (on page 173)

96325 CHANGE HISTORY

96326 First released in Issue 4.

96327 **Issue 7**
96328

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

96329 **NAME**

96330 logname — return the user's login name

96331 **SYNOPSIS**

96332 logname

96333 **DESCRIPTION**

96334 The *logname* utility shall write the user's login name to standard output. The login name shall be
 96335 the string that would be returned by the *getlogin()* function defined in the System Interfaces
 96336 volume of POSIX.1-2017. Under the conditions where the *getlogin()* function would fail, the
 96337 *logname* utility shall write a diagnostic message to standard error and exit with a non-zero exit
 96338 status.

96339 **OPTIONS**

96340 None.

96341 **OPERANDS**

96342 None.

96343 **STDIN**

96344 Not used.

96345 **INPUT FILES**

96346 None.

96347 **ENVIRONMENT VARIABLES**96348 The following environment variables shall affect the execution of *logname*:

96349 *LANG* Provide a default value for the internationalization variables that are unset or null.
 96350 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 96351 variables used to determine the values of locale categories.)

96352 *LC_ALL* If set to a non-empty string value, override the values of all the other
 96353 internationalization variables.

96354 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 96355 characters (for example, single-byte as opposed to multi-byte characters in
 96356 arguments).

96357 *LC_MESSAGES*

96358 Determine the locale that should be used to affect the format and contents of
 96359 diagnostic messages written to standard error.

96360 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

96361 **ASYNCHRONOUS EVENTS**

96362 Default.

96363 **STDOUT**96364 The *logname* utility output shall be a single line consisting of the user's login name:

96365 "%s\n", <login name>

96366 **STDERR**

96367 The standard error shall be used only for diagnostic messages.

96368 **OUTPUT FILES**

96369 None.

96370 **EXTENDED DESCRIPTION**

96371 None.

96372 **EXIT STATUS**

96373 The following exit values shall be returned:

96374 0 Successful completion.

96375 >0 An error occurred.

96376 **CONSEQUENCES OF ERRORS**

96377 Default.

96378 **APPLICATION USAGE**96379 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment
96380 changes could produce erroneous results.96381 **EXAMPLES**

96382 None.

96383 **RATIONALE**96384 The **passwd** file is not listed as required because the implementation may have other means of
96385 mapping login names.96386 **FUTURE DIRECTIONS**

96387 None.

96388 **SEE ALSO**96389 *id*, *who*96390 XBD [Chapter 8](#) (on page 173)96391 XSH [getlogin\(\)](#)96392 **CHANGE HISTORY**

96393 First released in Issue 2.

96394 **NAME**

96395 lp ‡send files to a printer

96396 **SYNOPSIS**96397 lp [-c] [-d *dest*] [-n *copies*] [-msw] [-o *option*]... [-t *title*] [*file*...]96398 **DESCRIPTION**

96399 The *lp* utility shall copy the input files to an output destination in an unspecified manner. The
 96400 default output destination should be to a hardcopy device, such as a printer or microfilm
 96401 recorder, that produces non-volatile, human-readable documents. If such a device is not
 96402 available to the application, or if the system provides no such device, the *lp* utility shall exit with
 96403 a non-zero exit status.

96404 The actual writing to the output device may occur some time after the *lp* utility successfully
 96405 exits. During the portion of the writing that corresponds to each input file, the implementation
 96406 shall guarantee exclusive access to the device.

96407 The *lp* utility shall associate a unique *request ID* with each request.

96408 Normally, a banner page is produced to separate and identify each print job. This page may be
 96409 suppressed by implementation-defined conditions, such as an operator command or one of the
 96410 **-o** *option* values.

96411 **OPTIONS**96412 The *lp* utility shall conform to XBD [Section 12.2](#) (on page 216).

96413 The following options shall be supported:

96414 **-c** Exit only after further access to any of the input files is no longer required. The
 96415 application can then safely delete or modify the files without affecting the output
 96416 operation. Normally, files are not copied, but are linked whenever possible. If the
 96417 **-c** option is not given, then the user should be careful not to remove any of the
 96418 files before the request has been printed in its entirety. It should also be noted that
 96419 in the absence of the **-c** option, any changes made to the named files after the
 96420 request is made but before it is printed may be reflected in the printed output. On
 96421 some implementations, **-c** may be on by default.

96422 **-d** *dest* Specify a string that names the destination (*dest*). If *dest* is a printer, the request
 96423 shall be printed only on that specific printer. If *dest* is a class of printers, the request
 96424 shall be printed on the first available printer that is a member of the class. Under
 96425 certain conditions (printer unavailability, file space limitation, and so on), requests
 96426 for specific destinations need not be accepted. Destination names vary between
 96427 systems.

96428 If **-d** is not specified, and neither the *LPDEST* nor *PRINTER* environment variable
 96429 is set, an unspecified destination is used. The **-d** *dest* option shall take precedence
 96430 over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are
 96431 undefined when *dest* contains a value that is not a valid destination name.

96432 **-m** Send mail (see [mailx](#)) after the files have been printed. By default, no mail is sent
 96433 upon normal completion of the print request.

96434 **-n** *copies* Write *copies* number of copies of the files, where *copies* is a positive decimal integer.
 96435 The methods for producing multiple copies and for arranging the multiple copies
 96436 when multiple *file* operands are used are unspecified, except that each file shall be
 96437 output as an integral whole, not interleaved with portions of other files.

- 96438 **-o option** Specify printer-dependent or class-dependent *options*. Several such *options* may be
96439 collected by specifying the **-o** option more than once.
- 96440 **-s** Suppress messages from *lp*.
- 96441 **-t title** Write *title* on the banner page of the output.
- 96442 **-w** Write a message on the user's terminal after the files have been printed. If the user
96443 is not logged in, then mail shall be sent instead.

96444 OPERANDS

96445 The following operand shall be supported:

- 96446 *file* A pathname of a file to be output. If no *file* operands are specified, or if a *file*
96447 operand is '-', the standard input shall be used. If a *file* operand is used, but the
96448 **-c** option is not specified, the process performing the writing to the output device
96449 may have user and group permissions that differ from that of the process invoking
96450 *lp*.

96451 STDIN

96452 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
96453 See the INPUT FILES section.

96454 INPUT FILES

96455 The input files shall be text files.

96456 ENVIRONMENT VARIABLES

96457 The following environment variables shall affect the execution of *lp*:

- 96458 **LANG** Provide a default value for the internationalization variables that are unset or null.
96459 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
96460 variables used to determine the values of locale categories.)

- 96461 **LC_ALL** If set to a non-empty string value, override the values of all the other
96462 internationalization variables.

- 96463 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
96464 characters (for example, single-byte as opposed to multi-byte characters in
96465 arguments and input files).

96466 LC_MESSAGES

- 96467 Determine the locale that should be used to affect the format and contents of
96468 diagnostic messages written to standard error and informative messages written to
96469 standard output.

- 96470 **LC_TIME** Determine the format and contents of date and time strings displayed in the *lp*
96471 banner page, if any.

- 96472 **LPDEST** Determine the destination. If the **LPDEST** environment variable is not set, the
96473 **PRINTER** environment variable shall be used. The **-d dest** option takes precedence
96474 over **LPDEST**. Results are undefined when **-d** is not specified and **LPDEST**
96475 contains a value that is not a valid destination name.

- 96476 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

- 96477 **PRINTER** Determine the output device or destination. If the **LPDEST** and **PRINTER**
96478 environment variables are not set, an unspecified output device is used. The **-d**
96479 **dest** option and the **LPDEST** environment variable shall take precedence over
96480 **PRINTER**. Results are undefined when **-d** is not specified, **LPDEST** is unset, and
96481 **PRINTER** contains a value that is not a valid device or destination name.

96482 *TZ* Determine the timezone used to calculate date and time strings displayed in the *lp*
96483 banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be
96484 used.

96485 **ASYNCHRONOUS EVENTS**

96486 Default.

96487 **STDOUT**

96488 The *lp* utility shall write a *request ID* to the standard output, unless *-s* is specified. The format of
96489 the message is unspecified. The request ID can be used on systems supporting the historical
96490 *cancel* and *lpstat* utilities.

96491 **STDERR**

96492 The standard error shall be used only for diagnostic messages.

96493 **OUTPUT FILES**

96494 None.

96495 **EXTENDED DESCRIPTION**

96496 None.

96497 **EXIT STATUS**

96498 The following exit values shall be returned:

96499 0 All input files were processed successfully.

96500 >0 No output device was available, or an error occurred.

96501 **CONSEQUENCES OF ERRORS**

96502 Default.

96503 **APPLICATION USAGE**

96504 The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's
96505 default page size.

96506 A conforming application can use one of the *file* operands only with the *-c* option or if the file is
96507 publicly readable and guaranteed to be available at the time of printing. This is because
96508 POSIX.1-2017 gives the implementation the freedom to queue up the request for printing at
96509 some later time by a different process that might not be able to access the file.

96510 **EXAMPLES**

96511 1. To print file *file*:

96512 `lp -c file`

96513 2. To print multiple files with headers:

96514 `pr file1 file2 | lp`

96515 **RATIONALE**

96516 The *lp* utility was designed to be a basic version of a utility that is already available in many
96517 historical implementations. The standard developers considered that it should be implementable
96518 simply as:

96519 `cat "$@" > /dev/lp`

96520 after appropriate processing of options, if that is how the implementation chose to do it and if
96521 exclusive access could be granted (so that two users did not write to the device simultaneously).
96522 Although in the future the standard developers may add other options to this utility, it should
96523 always be able to execute with no options or operands and send the standard input to an

96524 unspecified output device.

96525 This volume of POSIX.1-2017 makes no representations concerning the format of the printed
96526 output, except that it must be “human-readable” and “non-volatile”. Thus, writing by default to
96527 a disk or tape drive or a display terminal would not qualify. (Such destinations are not
96528 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

96529 This volume of POSIX.1-2017 is worded such that a “print job” consisting of multiple input files,
96530 possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with
96531 another, but there is no statement that all the files or copies have to print out together.

96532 The `-c` option may imply a spooling operation, but this is not required. The utility can be
96533 implemented to wait until the printer is ready and then wait until it is finished. Because of that,
96534 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

96535 On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel
96536 or find the status of a request using utilities not defined in this volume of POSIX.1-2017.

96537 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,
96538 they used different names for the environment variable specifying the destination printer. Since
96539 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence
96540 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one
96541 or the other environment variable, the `lp` utility is required to recognize both. If this was not
96542 done, many applications would send output to unexpected output devices when users moved
96543 from system to system.

96544 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests
96545 have proposed additional options or operands or both that added functionality. The requests
96546 included:

96547 Wording *requiring* the output to be “hardcopy”

96548 A requirement for multiple printers

96549 Options for supporting various page-description languages

96550 Given that a compliant system is not required to even have a printer, placing further restrictions
96551 upon the behavior of the printer is not useful. Since hardcopy format is so application-
96552 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that
96553 should be required on all compliant systems.

96554 The term *unspecified* is used in this section in lieu of *implementation-defined* as most known
96555 implementations would not be able to make definitive statements in their conformance
96556 documents; the existence and usage of printers is very dependent on how the system
96557 administrator configures each individual system.

96558 Since the default destination, device type, queuing mechanisms, and acceptable forms of input
96559 are all unspecified, usage guidelines for what a conforming application can do are as follows:

96560 Use the command in a pipeline, or with `-c`, so that there are no permission problems and
96561 the files can be safely deleted or modified.

96562 Limit output to text files of reasonable line lengths and printable characters and include no
96563 device-specific formatting information, such as a page description language. The meaning
96564 of “reasonable” in this context can only be answered as a quality-of-implementation issue,
96565 but it should be apparent from historical usage patterns in the industry and the locale. The
96566 `pr` and `fold` utilities can be used to achieve reasonable formatting for the default page size
96567 of the implementation.

96568 Alternatively, the application can arrange its installation in such a way that it requires the system
 96569 administrator or operator to provide the appropriate information on *lp* options and environment
 96570 variable values.

96571 At a minimum, having this utility in this volume of POSIX.1-2017 tells the industry that
 96572 conforming applications require a means to print output and provides at least a command name
 96573 and *LPDEST* routing mechanism that can be used for discussions between vendors, application
 96574 developers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent
 96575 of the standard developers, even if they cannot mandate that all systems (such as laptops) have
 96576 printers.

96577 This volume of POSIX.1-2017 does not specify what the ownership of the process performing the
 96578 writing to the output device may be. If *-c* is not used, it is unspecified whether the process
 96579 performing the writing to the output device has permission to read *file* if there are any
 96580 restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the
 96581 results of deleting *file* before it is printed are unspecified.

96582 FUTURE DIRECTIONS

96583 None.

96584 SEE ALSO

96585 *mailx*

96586 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

96587 CHANGE HISTORY

96588 First released in Issue 2.

96589 Issue 6

96590 The following new requirements on POSIX implementations derive from alignment with the
 96591 Single UNIX Specification:

96592 In the DESCRIPTION, the requirement to associate a unique request ID, and the normal
 96593 generation of a banner page is added.

96594 In the OPTIONS section:

96595 ‡ *lpstat -d dest* description is expanded, but references to *lpstat* are removed.

96596 ‡ *lpstat -m, -o, -s, -t, and -w* options are added.

96597 In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.

96598 The STDOUT section is added.

96599 The normative text is reworded to avoid use of the term “must” for application requirements.

96600 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

96601 Issue 7

96602 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

96603 **NAME**96604 `ls` — list directory contents96605 **SYNOPSIS**

```
96606 XSI ls [-ikqrs] [-g lno] [-A|-a] [-C|-m|-x|-l] \
96607 [-F|-p] [-H|-L] [-R|-d] [-S|-f|-t] [-c|-u] [file...]
```

96608 **DESCRIPTION**

96609 For each operand that names a file of a type other than directory or symbolic link to a directory,
 96610 *ls* shall write the name of the file as well as any requested, associated information. For each
 96611 operand that names a file of type directory, *ls* shall write the names of files contained within the
 96612 directory as well as any requested, associated information. Filenames beginning with a <period>
 96613 ('.') and any associated information shall not be written out unless explicitly referenced, the
 96614 `-A` or `-a` option is supplied, or an implementation-defined condition causes them to be written.
 96615 If one or more of the `-d`, `-F`, or `-l` options are specified, and neither the `-H` nor the `-L` option is
 96616 specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write the
 96617 name of the file as well as any requested, associated information. If none of the `-d`, `-F`, or `-l`
 96618 options are specified, or the `-H` or `-L` options are specified, for each operand that names a file of
 96619 type symbolic link to a directory, *ls* shall write the names of files contained within the directory
 96620 as well as any requested, associated information. In each case where the names of files contained
 96621 within a directory are written, if the directory contains any symbolic links then *ls* shall evaluate
 96622 the file information and file type to be those of the symbolic link itself, unless the `-L` option is
 96623 specified.

96624 If no operands are specified, *ls* shall behave as if a single operand of dot ('.') had been
 96625 specified. If more than one operand is specified, *ls* shall write non-directory operands first; it
 96626 shall sort directory and non-directory operands separately according to the collating sequence in
 96627 the current locale.

96628 Whenever *ls* sorts filenames or pathnames according to the collating sequence in the current
 96629 locale, if this collating sequence does not have a total ordering of all characters (see XBD [Section](#)
 96630 [7.3.2](#), on page 147), then any filenames or pathnames that collate equally should be further
 96631 compared byte-by-byte using the collating sequence for the POSIX locale.

96632 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an
 96633 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic
 96634 message to standard error and shall either recover its position in the hierarchy or terminate.

96635 **OPTIONS**96636 The *ls* utility shall conform to XBD [Section 12.2](#) (on page 216).

96637 The following options shall be supported:

96638 `-A` Write out all directory entries, including those whose names begin with a <period>
 96639 ('.') but excluding the entries dot and dot-dot (if they exist).

96640 `-C` Write multi-text-column output with entries sorted down the columns, according
 96641 to the collating sequence. The number of text columns and the column separator
 96642 characters are unspecified, but should be adapted to the nature of the output
 96643 device. This option disables long format output.

96644 `-F` Do not follow symbolic links named as operands unless the `-H` or `-L` options are
 96645 specified. Write a <slash> ('/') immediately after each pathname that is a
 96646 directory, an <asterisk> ('*') after each that is executable, a <vertical-line> ('|')
 96647 after each that is a FIFO, and an at-sign ('@') after each that is a symbolic link. For
 96648 other file types, other symbols may be written.

96649		-H	Evaluate the file information and file type for symbolic links specified on the command line to be those of the file referenced by the link, and not the link itself; however, <i>ls</i> shall write the name of the link itself and not the file referenced by the link.
96650			
96651			
96652			
96653		-L	Evaluate the file information and file type for all symbolic links (whether named on the command line or encountered in a file hierarchy) to be those of the file referenced by the link, and not the link itself; however, <i>ls</i> shall write the name of the link itself and not the file referenced by the link. When -L is used with -l , write the contents of symbolic links in the long format (see the STDOUT section).
96654			
96655			
96656			
96657			
96658		-R	Recursively list subdirectories encountered. When a symbolic link to a directory is encountered, the directory shall not be recursively listed unless the -L option is specified. The use of -R with -d or -f produces unspecified results.
96659			
96660			
96661		-S	Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).
96662			
96663		-a	Write out all directory entries, including those whose names begin with a <period> ('.').
96664			
96665		-c	Use time of last modification of the file status information (see XBD < sys/stat.h >) instead of last modification of the file itself for sorting (-t) or writing (-l).
96666			
96667		-d	Do not follow symbolic links named as operands unless the -H or -L options are specified. Do not treat directories differently than other types of files. The use of -d with -R or -f produces unspecified results.
96668			
96669			
96670		-f	List the entries in directory operands in the order they appear in the directory. The behavior for non-directory operands is unspecified. This option shall turn on -a . When -f is specified, any occurrences of the -r , -S , and -t options shall be ignored and any occurrences of the -A , -g , -l , -n , -o , and -s options may be ignored. The use of -f with -R or -d produces unspecified results.
96671			
96672			
96673	XSI		
96674			
96675	XSI	-g	Turn on the -l (ell) option, but disable writing the file's owner name or number. Disable the -C , -m , and -x options.
96676			
96677		-i	For each file, write the file's file serial number (see <i>stat()</i> in the System Interfaces volume of POSIX.1-2017).
96678			
96679		-k	Set the block size for the -s option and the per-directory block count written for the -l , -n , -s , -g , and -o options (see the STDOUT section) to 1 024 bytes.
96680	XSI		
96681		-l	(The letter ell.) Do not follow symbolic links named as operands unless the -H or -L options are specified. Write out in long format (see the STDOUT section). Disable the -C , -m , and -x options.
96682			
96683			
96684		-m	Stream output format; list pathnames across the page, separated by a <comma> character followed by a <space> character. Use a <newline> character as the list terminator and after the separator sequence when there is not room on a line for the next list entry. This option disables long format output.
96685			
96686			
96687			
96688		-n	Turn on the -l (ell) option, but when writing the file's owner or group, write the file's numeric UID or GID rather than the user or group name, respectively. Disable the -C , -m , and -x options.
96689			
96690			
96691	XSI	-o	Turn on the -l (ell) option, but disable writing the file's group name or number. Disable the -C , -m , and -x options.
96692			

- 96693 **-p** Write a <slash> (' / ') after each filename if that file is a directory.
- 96694 **-q** Force each instance of non-printable filename characters and <tab> characters to be written as the <question-mark> (' ? ') character. Implementations may provide this option by default if the output is to a terminal device.
- 96695
- 96696
- 96697 **-r** Reverse the order of the sort to get reverse collating sequence oldest first, or smallest file size first depending on the other options given.
- 96698
- 96699 **-s** Indicate the total number of file system blocks consumed by each file displayed. If the **-k** option is also specified, the block size shall be 1024 bytes; otherwise, the block size is implementation-defined.
- 96700
- 96701
- 96702 **-t** Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence. For a symbolic link, the time used as the sort key is that of the symbolic link itself, unless *ls* is evaluating its file information to be that of the file referenced by the link (see the **-H** and **-L** options).
- 96703
- 96704
- 96705
- 96706
- 96707 **-u** Use time of last access (see XBD <sys/stat.h>) instead of last modification of the file for sorting (**-t**) or writing (**-l**).
- 96708
- 96709 **-x** The same as **-C**, except that the multi-text-column output is produced with entries sorted across, rather than down, the columns. This option disables long format output.
- 96710
- 96711
- 96712 **-1** (The numeric digit one.) Force output to be one entry per line. This option does not disable long format output. (Long format output is enabled by **-g**, **-l** (ell), **-n**, and **-o**; and disabled by **-C**, **-m**, and **-x**.)
- 96713 XSI
- 96714 XSI
- 96715 XSI If an option that enables long format output (**-g**, **-l** (ell), **-n**, and **-o**) is given with an option that disables long format output (**-C**, **-m**, and **-x**), this shall not be considered an error. The last of these options specified shall determine whether long format output is written.
- 96716
- 96717
- 96718 If **-R**, **-d**, or **-f** are specified, the results of specifying these mutually-exclusive options are specified by the descriptions of these options above. If more than one of any of the other options shown in the SYNOPSIS section in mutually-exclusive sets are given, this shall not be considered an error; the last option specified in each set shall determine the output.
- 96719
- 96720
- 96721
- 96722 Note that if **-t** is specified, **-c** and **-u** are not only mutually-exclusive with each other, they are also mutually-exclusive with **-S** when determining sort order. But even if **-S** is specified after all occurrences of **-c**, **-t**, and **-u**, the last use of **-c** or **-u** determines the timestamp printed when producing long format output.
- 96723
- 96724
- 96725

96726 OPERANDS

96727 The following operand shall be supported:

- 96728 *file* A pathname of a file to be written. If the file specified is not found, a diagnostic message shall be output on standard error.
- 96729

96730 STDIN

96731 Not used.

96732 INPUT FILES

96733 None.

96734 **ENVIRONMENT VARIABLES**96735 The following environment variables shall affect the execution of *ls*:

96736 **COLUMNS** Determine the user's preferred column position width for writing multiple text-
 96737 column output. If this variable contains a string representing a decimal integer, the
 96738 *ls* utility shall calculate how many pathname text columns to write (see **-C**) based
 96739 on the width provided. If **COLUMNS** is not set or invalid, an implementation-
 96740 defined number of column positions shall be assumed, based on the
 96741 implementation's knowledge of the output device. The column width chosen to
 96742 write the names of files in any given directory shall be constant. Filenames shall
 96743 not be truncated to fit into the multiple text-column output.

96744 **LANG** Provide a default value for the internationalization variables that are unset or null.
 96745 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 96746 variables used to determine the values of locale categories.)

96747 **LC_ALL** If set to a non-empty string value, override the values of all the other
 96748 internationalization variables.

96749 **LC_COLLATE**
 96750 Determine the locale for character collation information in determining the
 96751 pathname collation sequence.

96752 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 96753 characters (for example, single-byte as opposed to multi-byte characters in
 96754 arguments) and which characters are defined as printable (character class **print**).

96755 **LC_MESSAGES**
 96756 Determine the locale that should be used to affect the format and contents of
 96757 diagnostic messages written to standard error.

96758 **LC_TIME** Determine the format and contents for date and time strings written by *ls*.

96759 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

96760 **TZ** Determine the timezone for date and time strings written by *ls*. If **TZ** is unset or
 96761 null, an unspecified default timezone shall be used.

96762 **ASYNCHRONOUS EVENTS**

96763 Default.

96764 **STDOUT**

96765 The default format shall be to list one entry per line to standard output; the exceptions are to
 96766 terminals or when one of the **-C**, **-m**, or **-x** options is specified. If the output is to a terminal, the
 96767 format is implementation-defined.

96768 When **-m** is specified, the format used for the last element of the list shall be:

96769 "%s\n", <filename>

96770 The format used for each other element of the list shall be:

96771 "%s,%s", <filename>, <separator>

96772 where, if there is not room for the next element of the list to fit within the current line length,
 96773 <separator> is a string containing an optional <space> character and a mandatory <newline>
 96774 character; otherwise it is a single <space> character.

96775 If the **-i** option is specified, the file's file serial number (see XBD [<sys/stat.h>](#)) shall be written in
 96776 the following format before any other output for the corresponding entry:

96777 %u ", <file serial number>

96778 If the **-l** option is specified, the following information shall be written for files other than
 96779 character special and block special files:

96780 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,
 96781 <owner name>, <group name>, <size>, <date and time>,
 96782 <pathname>

96783 If the **-l** option is specified, the following information shall be written for character special and
 96784 block special files:

96785 "%s %u %s %s %s %s %s\n", <file mode>, <number of links>,
 96786 <owner name>, <group name>, <device info>, <date and time>,
 96787 <pathname>

96788 In both cases if the file is a symbolic link and the **-L** option is also specified, this information
 96789 shall be for the file resolved from the symbolic link, except that the <pathname> field shall
 96790 contain the pathname of the symbolic link itself. If the file is a symbolic link and the **-L** option is
 96791 not specified, this information shall be about the link itself and the <pathname> field shall be of
 96792 the form:

96793 "%s -> %s", <pathname of link>, <contents of link>

96794 XSI The **-n**, **-g**, and **-o** options use the same format as **-l**, but with omitted items and their
 96795 associated <blank> characters. See the OPTIONS section.

96796 In both the preceding **-l** forms, if <owner name> or <group name> cannot be determined, or if **-n**
 96797 is given, they shall be replaced with their associated numeric values using the format %u.

96798 The <size> field shall contain the value that would be returned for the file in the *st_size* field of
 96799 **struct stat** (see XBD <[sys/stat.h](#)>). Note that for some file types this value is unspecified.

96800 The <device info> field shall contain implementation-defined information associated with the
 96801 device in question.

96802 The <date and time> field shall contain the appropriate date and timestamp of when the file was
 96803 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following
 96804 *date* command:

96805 date "+%b %e %H:%M"

96806 if the file has been modified in the last six months, or:

96807 date "+%b %e %Y"

96808 (where two <space> characters are used between %e and %Y) if the file has not been modified in
 96809 the last six months or if the modification date is in the future, except that, in both cases, the final
 96810 <newline> produced by *date* shall not be included and the output shall be as if the *date*
 96811 command were executed at the time of the last modification date of the file rather than the
 96812 current time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format
 96813 and order of presentation of this field may be used.

96814 If the pathname was specified as a *file* operand, it shall be written as specified.

96815 XSI The file mode written under the **-l**, **-n**, **-g**, and **-o** options shall consist of the following format:

96816 "%c%s%s%s", <entry type>, <owner permissions>,
 96817 <group permissions>, <other permissions>,
 96818 <optional alternate access method flag>

96819 The *<optional alternate access method flag>* shall be the empty string if there is no alternate or
 96820 additional access control method associated with the file; otherwise, it shall be a string
 96821 containing a single printable character that is not a <blank>.

96822 The *<entry type>* character shall describe the type of file, as follows:

- 96823 d Directory.
- 96824 b Block special file.
- 96825 c Character special file.
- 96826 l (ell) Symbolic link.
- 96827 p FIFO.
- 96828 – Regular file.

96829 Implementations may add other characters to this list to represent other implementation-defined
 96830 file types.

96831 The next three fields shall be three characters each:

96832 *<owner permissions>*

96833 Permissions for the file owner class (see XBD [Section 4.5](#), on page 108).

96834 *<group permissions>*

96835 Permissions for the file group class.

96836 *<other permissions>*

96837 Permissions for the file other class.

96838 Each field shall have three character positions:

- 96839 1. If 'r', the file is readable; if '–', the file is not readable.
- 96840 2. If 'w', the file is writable; if '–', the file is not writable.
- 96841 3. The first of the following that applies:
 - 96842 S If in *<owner permissions>*, the file is not executable and set-user-ID mode is set. If in
 96843 *<group permissions>*, the file is not executable and set-group-ID mode is set.
 - 96844 s If in *<owner permissions>*, the file is executable and set-user-ID mode is set. If in
 96845 *<group permissions>*, the file is executable and set-group-ID mode is set.
 - 96846 XSI T If in *<other permissions>* and the file is a directory, search permission is not granted to
 96847 others, and the restricted deletion flag is set.
 - 96848 XSI t If in *<other permissions>* and the file is a directory, search permission is granted to
 96849 others, and the restricted deletion flag is set.
 - 96850 x The file is executable or the directory is searchable.
 - 96851 – None of the attributes of 'S', 's', 'T', 't', or 'x' applies.

96852 Implementations may add other characters to this list for the third character position.
 96853 Such additions shall, however, be written in lowercase if the file is executable or
 96854 searchable, and in uppercase if it is not.

96855 XSI If any of the **-l**, **-n**, **-s**, **-g**, or **-o** options is specified, each list of files within the directory shall be
 96856 preceded by a status line indicating the number of file system blocks occupied by files in the
 96857 directory in 512-byte units if the **-k** option is not specified, or 1 024-byte units if the **-k** option is
 96858 specified, rounded up to the next integral number of units, if necessary. In the POSIX locale, the

96859 format shall be:

96860 "total %u\n", <number of units in the directory>

96861 If more than one directory, or a combination of non-directory files and directories are written,
96862 either as a result of specifying multiple operands, or the **-R** option, each list of files within a
96863 directory shall be preceded by:

96864 "\n%s:\n", <directory name>

96865 If this string is the first thing to be written, the first <newline> shall not be written. This output
96866 shall precede the number of units in the directory.

96867 If the **-s** option is given, each file shall be written with the number of blocks used by the file.
96868 XSI Along with **-C**, **-l**, **-m**, or **-x**, the number and a <space> shall precede the filename; with **-l**, **-n**,
96869 **-g**, or **-o**, they shall precede each line describing a file.

96870 **STDERR**

96871 The standard error shall be used only for diagnostic messages.

96872 **OUTPUT FILES**

96873 None.

96874 **EXTENDED DESCRIPTION**

96875 None.

96876 **EXIT STATUS**

96877 The following exit values shall be returned:

96878 0 Successful completion.

96879 >0 An error occurred.

96880 **CONSEQUENCES OF ERRORS**

96881 Default.

96882 **APPLICATION USAGE**

96883 Many implementations use the <equals-sign> ('=') to denote sockets bound to the file system
96884 for the **-F** option. Similarly, many historical implementations use the 's' character to denote
96885 sockets as the entry type characters for the **-l** option.

96886 It is difficult for an application to use every part of the file modes field of *ls -l* in a portable
96887 manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as
96888 implementations may have extensions. Applications can use this field to pass directly to a user
96889 printout or prompt, but actions based on its contents should generally be deferred, instead, to
96890 the *test* utility.

96891 The output of *ls* (with the **-l** and related options) contains information that logically could be
96892 used by utilities such as *chmod* and *touch* to restore files to a known state. However, this
96893 information is presented in a format that cannot be used directly by those utilities or be easily
96894 translated into a format that can be used. A character has been added to the end of the
96895 permissions string so that applications at least have an indication that they may be working in
96896 an area they do not understand instead of assuming that they can translate the permissions
96897 string into something that can be used. Future versions or related documents may define one or
96898 more specific characters to be used based on different standard additional or alternative access
96899 control mechanisms.

96900 As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one
96901 of the long listing formats must be used carefully on systems where filenames can contain
96902 embedded white space. Systems and system administrators should institute policies and user

96903 training to limit the use of such filenames.

96904 The number of disk blocks occupied by the file that it reports varies depending on underlying
 96905 file system type, block size units reported, and the method of calculating the number of blocks.
 96906 On some file system types, the number is the actual number of blocks occupied by the file
 96907 (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the
 96908 file size (usually making an allowance for indirect blocks, but ignoring holes).

96909 EXAMPLES

96910 An example of a small directory tree being fully listed with `ls -laRF a` in the POSIX locale:

```
96911 total 11
96912 drwxr-xr-x  3 fox    prog      64 Jul  4 12:07 ./
96913 drwxrwxrwx  4 fox    prog    3264 Jul  4 12:09 ../
96914 drwxr-xr-x  2 fox    prog      48 Jul  4 12:07 b/
96915 -rwxr--r--  1 fox    prog     572 Jul  4 12:07 foo*
```

96916 a/b:

```
96917 total 4
96918 drwxr-xr-x  2 fox    prog      48 Jul  4 12:07 ./
96919 drwxr-xr-x  3 fox    prog      64 Jul  4 12:07 ../
96920 -rw-r--r--  1 fox    prog     700 Jul  4 12:07 bar
```

96921 RATIONALE

96922 Some historical implementations of the `ls` utility show all entries in a directory except dot and
 96923 dot-dot when a superuser invokes `ls` without specifying the `-a` option. When “normal” users
 96924 invoke `ls` without specifying `-a`, they should not see information about any files with names
 96925 beginning with a `<period>` unless they were named as *file* operands.

96926 Implementations are expected to traverse arbitrary depths when processing the `-R` option. The
 96927 only limitation on depth should be based on running out of physical storage for keeping track of
 96928 untraversed directories.

96929 The `-1` (one) option was historically found in BSD and BSD-derived implementations only. It is
 96930 required in this volume of POSIX.1-2017 so that conforming applications might ensure that
 96931 output is one entry per line, even if the output is to a terminal.

96932 The `-S` option was added in Issue 7, but had been provided by several implementations for
 96933 many years. The description given in the standard documents historic practice, but does not
 96934 match much of the documentation that described its behavior. Historical documentation
 96935 typically described it as something like:

96936 `-S` Sort by size (largest size first) instead of by name. Special character devices (listed
 96937 last) are sorted by name.

96938 even though the file type was never considered when sorting the output. Character special files
 96939 do typically sort close to the end of the list because their file size on most implementations is
 96940 zero. But they are sorted alphabetically with any other files that happen to have the same file
 96941 size (zero), not sorted separately and added to the end.

96942 This volume of POSIX.1-2017 is frequently silent about what happens when mutually-exclusive
 96943 options are specified. Except for `-R`, `-d`, and `-f`, the `ls` utility is required to accept multiple
 96944 options from each mutually-exclusive option set without treating them as errors and to use the
 96945 behavior specified by the last option given in each mutually-exclusive set. Since `ls` is one of the
 96946 most aliased commands, it is important that the implementation perform intuitively. For
 96947 example, if the alias were:

```
96948 alias ls="ls -C"
```

96949 and the user typed `ls -1` (one), single-text-column output should result, not an error.

96950 The `-g`, `-l` (ell), `-n`, and `-o` options are not mutually-exclusive options. They all enable long
 96951 format output. They work together to determine whether the file's owner is written (no if `-g` is
 96952 present), file's group is written (no if `-o` is present), and if the file's group or owner is written
 96953 whether it is written as the name (default) or a string representation of the UID or GID number
 96954 (if `-n` is present). The `-C`, `-m`, `-x`, and `-1` (one) are mutually-exclusive options and the first three
 96955 of these disable long format output. The `-1` (one) option does not directly change whether or not
 96956 long format output is enabled, but by overriding `-C`, `-m`, and `-x`, it can re-enable long format
 96957 output that had been disabled by one of these options.

96958 Earlier versions of this standard did not describe the BSD `-A` option (like `-a`, but dot and dot-dot
 96959 are not written out). It has been added due to widespread implementation.

96960 Implementations may make `-q` the default for terminals to prevent trojan horse attacks on
 96961 terminals with special escape sequences. This is not required because:

96962 Some control characters may be useful on some terminals; for example, a system might
 96963 write them as `"\001"` or `"^A"`.

96964 Special behavior for terminals is not relevant to applications portability.

96965 An early proposal specified that the *<optional alternate access method flag>* had to be '+' if there
 96966 was an alternate access method used on the file or *<space>* if there was not. This was changed to
 96967 be *<space>* if there is not and a single printable character if there is. This was done for three
 96968 reasons:

- 96969 1. There are historical implementations using characters other than '+'.
- 96970 2. There are implementations that vary this character used in that position to distinguish
 96971 between various alternate access methods in use.
- 96972 3. The standard developers did not want to preclude future specifications that might need a
 96973 way to specify more than one alternate access method.

96974 Nonetheless, implementations providing a single alternate access method are encouraged to use
 96975 '+'.

96976 Earlier versions of this standard did not have the `-k` option, which meant that the `-s` option
 96977 could not be used portably as its block size was implementation-defined, and the units used to
 96978 specify the number of blocks occupied by files in a directory in an `ls -l` listing were fixed as
 96979 512-byte units. The `-k` option has been added to provide a way for the `-s` option to be used
 96980 portably, and for consistency it also changes the aforementioned units from 512-byte to
 96981 1024-byte.

96982 The *<date and time>* field in the `-l` format is specified only for the POSIX locale. As noted, the
 96983 format can be different in other locales. No mechanism for defining this is present in this volume
 96984 of POSIX.1-2017, as the appropriate vehicle is a messaging system; that is, the format should be
 96985 specified as a "message".

96986 FUTURE DIRECTIONS

96987 Allowing `-f` to ignore the `-A`, `-g`, `-l`, `-n`, `-o`, and `-s` options may be removed in a future version.

96988 A future version of this standard may require that if the collating sequence for the current locale
 96989 does not have a total ordering of all characters, any filenames or pathnames that collate equally
 96990 are further compared byte-by-byte using the collating sequence for the POSIX locale.

96991 **SEE ALSO**96992 *chmod*, *find*96993 XBD [Section 7.3.2](#) (on page 147), [Section 4.5](#) (on page 108), [Chapter 8](#) (on page 173), [Section 12.2](#)
96994 (on page 216), [<sys/stat.h>](#)96995 XSH *fstatat()*96996 **CHANGE HISTORY**

96997 First released in Issue 2.

96998 **Issue 5**

96999 A second FUTURE DIRECTION is added.

97000 **Issue 6**97001 The following new requirements on POSIX implementations derive from alignment with the
97002 Single UNIX Specification:97003 In the **-F** option, other symbols are allowed for other file types.

97004 Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

97005 The Open Group Base Resolution bwg2001-010 is applied, adding the **T** and **t** fields as part of
97006 the XSI option.97007 **Issue 7**97008 Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access
97009 method flag in the STDOUT section.97010 Austin Group Interpretation 1003.1-2001 #128 is applied, clarifying the DESCRIPTION and the
97011 definition of the **-R** option.97012 Austin Group Interpretation 1003.1-2001 #129 is applied, clarifying the behavior of *ls* when no
97013 operands are specified.97014 Austin Group Interpretation 1003.1-2001 #198 is applied, clarifying the requirements for the **-H**
97015 option.97016 SD5-XCU-ERN-50 is applied, adding the **-A** option.

97017 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97018 The **-S** option is added from The Open Group Technical Standard, 2006, Extended API Set
97019 Part 1.97020 The **-f**, **-m**, **-n**, **-p**, **-s**, and **-x** options are moved from the XSI option to the Base.97021 The description of the **-f**, **-s**, and **-t** options are revised and the **-k** option is added.97022 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0098 [424], XCU/TC1-2008/0099
97023 [424], XCU/TC1-2008/0100 [424], XCU/TC1-2008/0101 [424], XCU/TC1-2008/0102 [424],
97024 XCU/TC1-2008/0103 [424], XCU/TC1-2008/0104 [424], XCU/TC1-2008/0105 [423,424],
97025 XCU/TC1-2008/0106 [424], XCU/TC1-2008/0107 [424], XCU/TC1-2008/0108 [424],
97026 XCU/TC1-2008/0109 [424], XCU/TC1-2008/0110 [424], XCU/TC1-2008/0111 [423],
97027 XCU/TC1-2008/0112 [117], XCU/TC1-2008/0113 [117], XCU/TC1-2008/0114 [117],
97028 XCU/TC1-2008/0115 [424], and XCU/TC1-2008/0116 [424] are applied.97029 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0115 [963] and XCU/TC2-2008/0116
97030 [963] are applied.

97031 **NAME**

97032 m4 ‡'ma⦿ processor

97033 **SYNOPSIS**97034 m4 [-s] [-D *name*[=*val*]]... [-U *name*]... *file*...97035 **DESCRIPTION**97036 The *m4* utility is a macro processor that shall read one or more text files, process them according
97037 to their included macro statements, and write the results to standard output.97038 **OPTIONS**97039 The *m4* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the order of the **-D**
97040 and **-U** options shall be significant, and options can be interspersed with operands.

97041 The following options shall be supported:

97042 **-s** Enable line synchronization output for the *c99* preprocessor phase (that is, **#line**
97043 directives).97044 **-D *name*[=*val*]**97045 Define *name* to *val* or to null if *=val* is omitted.97046 **-U *name*** Undefine *name*.97047 **OPERANDS**

97048 The following operand shall be supported:

97049 *file* A pathname of a text file to be processed. If no *file* is given, or if it is '-', the
97050 standard input shall be read.97051 **STDIN**97052 The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.97053 **INPUT FILES**97054 The input file named by the *file* operand shall be a text file.97055 **ENVIRONMENT VARIABLES**97056 The following environment variables shall affect the execution of *m4*:97057 **LANG** Provide a default value for the internationalization variables that are unset or null.
97058 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
97059 variables used to determine the values of locale categories.)97060 **LC_ALL** If set to a non-empty string value, override the values of all the other
97061 internationalization variables.97062 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
97063 characters (for example, single-byte as opposed to multi-byte characters in
97064 arguments and input files).97065 **LC_MESSAGES**97066 Determine the locale that should be used to affect the format and contents of
97067 diagnostic messages written to standard error.97068 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.97069 **ASYNCHRONOUS EVENTS**

97070 Default.

97071 **STDOUT**

97072 The standard output shall be the same as the input files, after being processed for macro
97073 expansion.

97074 **STDERR**

97075 The standard error shall be used to display strings with the **errprint** macro, macro tracing
97076 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for
97077 diagnostic messages.

97078 **OUTPUT FILES**

97079 None.

97080 **EXTENDED DESCRIPTION**

97081 The *m4* utility shall compare each token from the input against the set of built-in and user-
97082 defined macros. If the token matches the name of a macro, then the token shall be replaced by
97083 the macro's defining text, if any, and rescanned for matching macro names. Once no portion of
97084 the token matches the name of a macro, it shall be written to standard output. Macros may have
97085 arguments, in which case the arguments shall be substituted into the defining text before it is
97086 rescanned.

97087 Macro calls have the form:

```
97088 name(arg1, arg2, ..., argn)
```

97089 Macro names shall consist of letters, digits, and underscores, where the first character is not a
97090 digit. Tokens not of this form shall not be treated as macros.

97091 The application shall ensure that the <left-parenthesis> immediately follows the name of the
97092 macro. If a token matching the name of a macro is not followed by a <left-parenthesis>, it is
97093 handled as a use of that macro without arguments.

97094 If a macro name is followed by a <left-parenthesis>, its arguments are the <comma>-separated
97095 tokens between the <left-parenthesis> and the matching <right-parenthesis>. Unquoted white-
97096 space characters preceding each argument shall be ignored. All other characters, including
97097 trailing white-space characters, are retained. <comma> characters enclosed between <left-
97098 parenthesis> and <right-parenthesis> characters do not delimit arguments.

97099 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be
97100 replaced by the first argument. Systems shall support at least nine arguments; only the first nine
97101 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with
97102 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The
97103 string "\$*" is replaced by a list of all of the arguments, separated by <comma> characters. The
97104 string "\$@" is replaced by a list of all of the arguments separated by <comma> characters, and
97105 each argument is quoted using the current left and right quoting strings. The string "\${"
97106 produces unspecified behavior.

97107 If fewer arguments are supplied than are in the macro definition, the omitted arguments are
97108 taken to be null. It is not an error if more arguments are supplied than are in the macro
97109 definition.

97110 No special meaning is given to any characters enclosed between matching left and right quoting
97111 strings, but the quoting strings are themselves discarded. By default, the left quoting string
97112 consists of a grave accent (backquote) and the right quoting string consists of an acute accent
97113 (single-quote); see also the **changequote** macro.

97114 Comments are written but not scanned for matching macro names; by default, the begin-
97115 comment string consists of the <number-sign> character and the end-comment string consists of
97116 a <newline>. See also the **changecom** and **dnl** macros.

97117 The *m4* utility shall make available the following built-in macros. They can be redefined, but
97118 once this is done the original meaning is lost. Their values shall be null unless otherwise stated.
97119 In the descriptions below, the term *defining text* refers to the value of the macro: the second
97120 argument to the **define** macro, among other things. Except for the first argument to the **eval**
97121 macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The
97122 string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval**
97123 built-in macros shall be in the form of a decimal-constant as defined in the C language.

97124 **changecom** The **changecom** macro shall set the begin-comment and end-comment strings.
97125 With no arguments, the comment mechanism shall be disabled. With a single non-
97126 null argument, that argument shall become the begin-comment and the <newline>
97127 shall become the end-comment string. With two non-null arguments, the first
97128 argument shall become the begin-comment string and the second argument shall
97129 become the end-comment string. The behavior is unspecified if either argument is
97130 provided but null. Systems shall support comment strings of at least five
97131 characters.

97132 **changequote** The **changequote** macro shall set the begin-quote and end-quote strings. With no
97133 arguments, the quote strings shall be set to the default values (that is, ` `'). The
97134 behavior is unspecified if there is a single argument or either argument is null.
97135 With two non-null arguments, the first argument shall become the begin-quote
97136 string and the second argument shall become the end-quote string. Systems shall
97137 support quote strings of at least five characters.

97138 **decr** The defining text of the **decr** macro shall be its first argument decremented by 1. It
97139 shall be an error to specify an argument containing any non-numeric characters.
97140 The behavior is unspecified if **decr** is not immediately followed by a <left-
97141 parenthesis>.

97142 **define** The second argument shall become the defining text of the macro whose name is
97143 the first argument. It is unspecified whether the **define** macro deletes all prior
97144 definitions of the macro named by its first argument or preserves all but the
97145 current definition of the macro. The behavior is unspecified if **define** is not
97146 immediately followed by a <left-parenthesis>.

97147 **defn** The defining text of the **defn** macro shall be the quoted definition (using the
97148 current quoting strings) of its arguments. The behavior is unspecified if **defn** is not
97149 immediately followed by a <left-parenthesis>.

97150 **divert** The *m4* utility maintains nine temporary buffers, numbered 1 to 9, inclusive.
97151 When the last of the input has been processed, any output that has been placed in
97152 these buffers shall be written to standard output in buffer-numerical order. The
97153 **divert** macro shall divert future output to the buffer specified by its argument.
97154 Specifying no argument or an argument of 0 shall resume the normal output
97155 process. Output diverted to a stream with a negative number shall be discarded.
97156 Behavior is implementation-defined if a stream number larger than 9 is specified. It
97157 shall be an error to specify an argument containing any non-numeric characters.

97158 **divnum** The defining text of the **divnum** macro shall be the number of the current output
97159 stream as a string.

97160 **dnl** The **dnl** macro shall cause *m4* to discard all input characters up to and including
97161 the next <newline>.

97162 97163	dumpdef	The dumpdef macro shall write the defined text to standard error for each of the macros specified as arguments, or, if no arguments are specified, for all macros.
97164 97165	errprint	The errprint macro shall write its arguments to standard error. The behavior is unspecified if errprint is not immediately followed by a <left-parenthesis>.
97166 97167 97168 97169	eval	The eval macro shall evaluate its first argument as an arithmetic expression, using signed integer arithmetic with at least 32-bit precision. At least the following C-language operators shall be supported, with precedence, associativity, and behavior as described in Section 1.1.2.1 (on page 2331):
97170 97171 97172 97173		() unary + unary - ~
97174 97175 97176 97177		! binary * / %
97178 97179 97180 97181 97182 97183 97184 97185 97186 97187		binary + binary - << >> < <= > >= == !=
97188 97189 97190 97191 97192		binary & ^ &&
97193 97194 97195 97196 97197 97198 97199 97200		Systems shall support octal and hexadecimal numbers as in the ISO C standard. The second argument, if specified, shall set the radix for the result; if the argument is blank or unspecified, the default is 10. Behavior is unspecified if the radix falls outside the range 2 to 36, inclusive. The third argument, if specified, sets the minimum number of digits in the result. Behavior is unspecified if the third argument is less than zero. It shall be an error to specify the second or third argument containing any non-numeric characters. The behavior is unspecified if eval is not immediately followed by a <left-parenthesis>.
97201 97202 97203 97204	ifdef	If the first argument to the ifdef macro is defined, the defining text shall be the second argument. Otherwise, the defining text shall be the third argument, if specified, or the null string, if not. The behavior is unspecified if ifdef is not immediately followed by a <left-parenthesis>.
97205 97206 97207 97208 97209	ifelse	The ifelse macro takes three or more arguments. If the first two arguments compare as equal strings (after macro expansion of both arguments), the defining text shall be the third argument. If the first two arguments do not compare as equal strings and there are three arguments, the defining text shall be null. If the first two arguments do not compare as equal strings and there are four or five arguments,

97210		the defining text shall be the fourth argument. If the first two arguments do not
97211		compare as equal strings and there are six or more arguments, the first three
97212		arguments shall be discarded and processing shall restart with the remaining
97213		arguments. The behavior is unspecified if ifelse is not immediately followed by a
97214		<left-parenthesis>.
97215	include	The defining text for the include macro shall be the contents of the file named by
97216		the first argument. It shall be an error if the file cannot be read. The behavior is
97217		unspecified if include is not immediately followed by a <left-parenthesis>.
97218	incr	The defining text of the incr macro shall be its first argument incremented by 1. It
97219		shall be an error to specify an argument containing any non-numeric characters.
97220		The behavior is unspecified if incr is not immediately followed by a <left-
97221		parenthesis>.
97222	index	The defining text of the index macro shall be the first character position (as a
97223		string) in the first argument where a string matching the second argument begins
97224		(zero origin), or -1 if the second argument does not occur. The behavior is
97225		unspecified if index is not immediately followed by a <left-parenthesis>.
97226	len	The defining text of the len macro shall be the length (as a string) of the first
97227		argument. The behavior is unspecified if len is not immediately followed by a
97228		<left-parenthesis>.
97229	m4exit	Exit from the <i>m4</i> utility. If the first argument is specified, it shall be the exit code. If
97230		no argument is specified, the exit code shall be zero. It shall be an error to specify
97231		an argument containing any non-numeric characters. If the first argument is zero
97232		or no argument is specified, and an error has previously occurred (for example, a
97233		<i>file</i> operand that could not be opened), it is unspecified whether the exit status is
97234		zero or non-zero.
97235	m4wrap	The first argument shall be processed when EOF is reached. If the m4wrap macro
97236		is used multiple times, the arguments specified shall be processed in the order in
97237		which the m4wrap macros were processed. The behavior is unspecified if m4wrap
97238		is not immediately followed by a <left-parenthesis>.
97239	OB maketemp	The defining text shall be the first argument, with any trailing 'x' characters
97240		replaced with the current process ID as a string. The behavior is unspecified if
97241		maketemp is not immediately followed by a <left-parenthesis>.
97242	mkstemp	The defining text shall be as if it were the resulting pathname after a successful call
97243		to the <i>mkstemp</i> () function defined in the System Interfaces volume of POSIX.1-2017
97244		called with the first argument to the macro invocation. If a file is created, that file
97245		shall be closed. If a file could not be created, the <i>m4</i> utility shall write a diagnostic
97246		message to standard error and shall continue processing input but its final exit
97247		status shall be non-zero; the defining text of the macro shall be the empty string.
97248		The behavior is unspecified if mkstemp is not immediately followed by a <left-
97249		parenthesis>.
97250	popdef	The popdef macro shall delete the current definition of its arguments, replacing
97251		that definition with the previous one. If there is no previous definition, the macro
97252		is undefined. The behavior is unspecified if popdef is not immediately followed by
97253		a <left-parenthesis>.
97254	pushdef	The pushdef macro shall be equivalent to the define macro with the exception that
97255		it shall preserve any current definition for future retrieval using the popdef macro.
97256		The behavior is unspecified if pushdef is not immediately followed by a <left-

97257		parenthesis>.
97258	shift	The defining text for the shift macro shall be a comma-separated list of its arguments except the first one. Each argument shall be quoted using the current quoting strings. The behavior is unspecified if shift is not immediately followed by a <left-parenthesis>.
97259		
97260		
97261		
97262	sinclude	The sinclude macro shall be equivalent to the include macro, except that it shall not be an error if the file is inaccessible. The behavior is unspecified if sinclude is not immediately followed by a <left-parenthesis>.
97263		
97264		
97265	substr	The defining text for the substr macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. The behavior is unspecified if substr is not immediately followed by a <left-parenthesis>.
97266		
97267		
97268		
97269		
97270		
97271		
97272		
97273		
97274	syscmd	The syscmd macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. The string result shall not be rescanned for macros while setting the defining text. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the sysval macro. The behavior is unspecified if syscmd is not immediately followed by a <left-parenthesis>.
97275		
97276		
97277		
97278		
97279		
97280	sysval	The defining text of the sysval macro shall be the exit value of the utility last invoked by the syscmd macro (as a string).
97281		
97282	traceon	The traceon macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.
97283		
97284		
97285	traceoff	The traceoff macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.
97286		
97287	translit	The defining text of the translit macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument. If no replacement character is specified for some source character because the second argument is longer than the third argument, that character shall be deleted from the first argument in translit 's defining text. The behavior is unspecified if the '-' character appears within the second or third argument anywhere besides the first or last character. The behavior is unspecified if the same character appears more than once in the second argument. The behavior is unspecified if translit is not immediately followed by a <left-parenthesis>.
97288		
97289		
97290		
97291		
97292		
97293		
97294		
97295		
97296		
97297	undefine	The undefine macro shall delete all definitions (including those preserved using the pushdef macro) of the macros named by its arguments. The behavior is unspecified if undefine is not immediately followed by a <left-parenthesis>.
97298		
97299		
97300	undivert	The undivert macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. The behavior is unspecified if an argument
97301		
97302		
97303		

97304 contains any non-numeric characters.

97305 **EXIT STATUS**

97306 The following exit values shall be returned:

97307 0 Successful completion.

97308 >0 An error occurred

97309 If the **m4exit** macro is used, the exit value can be specified by the input file.

97310 **CONSEQUENCES OF ERRORS**

97311 Default.

97312 **APPLICATION USAGE**

97313 The **defn** macro is useful for renaming macros, especially built-ins.

97314 Since **eval** defers to the ISO C standard, some operations have undefined behavior. In some
 97315 implementations, division or remainder by zero cause a fatal signal, even if the division occurs
 97316 on the short-circuited branch of "&&" or "||". Any operation that overflows in signed
 97317 arithmetic produces undefined behavior. Likewise, using the **shift** operators with a shift amount
 97318 that is not positive and smaller than the precision is undefined, as is shifting a negative number
 97319 to the right. Historically, not all implementations obeyed C-language precedence rules: '~' and
 97320 '!' were lower than '=='; '==' and '!=' were not lower than '<'; and '|' was not lower
 97321 than '^'; the liberal use of "(" can force the desired precedence even with these non-
 97322 compliant implementations. Furthermore, some traditional implementations treated '^' as an
 97323 exponentiation operator, although most implementations now use "***" as an extension for this
 97324 purpose.

97325 When a macro has been multiply defined via the **pushdef** macro, it is unspecified whether the
 97326 **define** macro will alter only the most recent definition (as though by **popdef** and **pushdef**), or
 97327 replace the entire stack of definitions with a single definition (as though by **undefine** and
 97328 **pushdef**). An application desiring particular behavior for the **define** macro in this case can
 97329 redefine it accordingly.

97330 Applications should use the **mkstemp** macro instead of the obsolescent **maketemp** macro for
 97331 creating temporary files.

97332 **EXAMPLES**

97333 If the file **m4src** contains the lines:

```
97334 The value of `VER' is "VER".
97335 #ifdef `VER', `VER' is defined to be VER., VER is not defined.
97336 #ifndef VER, 1, `VER' is `VER'.
97337 #ifndef VER, 2, `VER' is `VER'., `VER' is not 2.
97338 #endif
```

97339 then the command

97340 `m4 m4src`

97341 or the command:

97342 `m4 -U VER m4src`

97343 produces the output:

```
97344 The value of VER is "VER".
97345 VER is not defined.
97346 VER is not 2.
```

```

97347     end
97348     The command:
97349     m4 -D VER m4src
97350     produces the output:
97351     The value of VER is "".
97352     VER is defined to be .
97353     VER is not 2.
97354     end
97355     The command:
97356     m4 -D VER=1 m4src
97357     produces the output:
97358     The value of VER is "1".
97359     VER is defined to be 1.
97360     VER is 1.
97361     VER is not 2.
97362     end
97363     The command:
97364     m4 -D VER=2 m4src
97365     produces the output:
97366     The value of VER is "2".
97367     VER is defined to be 2.
97368     VER is 2.
97369     end

```

97370 RATIONALE

97371 Historic System V-based behavior treated "\${" in a macro definition as two literal characters.
 97372 However, this sequence is left unspecified so that implementations may offer extensions such as
 97373 "\${11}" meaning the eleventh positional parameter. Macros can still be defined with
 97374 appropriate uses of nested quoting to result in a literal "\${" in the output after rescanning
 97375 removes the nested quotes.

97376 In the **translit** built-in, historic System V-based behavior treated '-' as a literal; GNU behavior
 97377 treats it as a range. This version of the standard allows either behavior.

97378 FUTURE DIRECTIONS

97379 None.

97380 SEE ALSO

97381 [c99](#)

97382 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

97383 CHANGE HISTORY

97384 First released in Issue 2.

97385 **Issue 5**

97386 The phrase “the defined text for macros written by the **dumpdef** macro” is added to the
97387 description of STDERR, and the description of **dumpdef** is updated to indicate that output is
97388 written to standard error. The description of **eval** is updated to indicate that the list of excluded
97389 C operators excludes unary '&' and '.'. In the description of **ifdef**, the phrase “and it is not
97390 defined to be zero” is deleted.

97391 **Issue 6**

97392 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a '&' character in the
97393 excepted list.

97394 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion
97395 buffers.

97396 The normative text is reworded to avoid use of the term “must” for application requirements.

97397 The Open Group Base Resolution bwg2000-006 is applied.

97398 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES
97399 section.

97400 **Issue 7**

97401 Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro
97402 obsolescent and adding a new **mkstemp** macro.

97403 Austin Group Interpretation 1003.1-2001 #207 is applied, clarifying the handling of white-space
97404 characters that precede or trail any macro arguments.

97405 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
97406 apply (options can be interspersed with operands).

97407 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97408 SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED
97409 DESCRIPTION.

97410 SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED
97411 DESCRIPTION.

97412 SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED
97413 DESCRIPTION.

97414 SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string "\$ {"
97415 produces unspecified behavior.

97416 SD5-XCU-ERN-112 is applied, updating the **changequote** macro.

97417 SD5-XCU-ERN-118 is applied, clarifying the definition of the **define** macro in the EXTENDED
97418 DESCRIPTION and APPLICATION USAGE sections.

97419 SD5-XCU-ERN-119 is applied, clarifying the definition of the **translit** macro in the EXTENDED
97420 DESCRIPTION and RATIONALE sections.

97421 SD5-XCU-ERN-130, SD5-XCU-ERN-131, and SD5-XCU-ERN-137 are applied.

97422 The *m4* utility is moved from the XSI option to the Base.

97423 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0117 [241], XCU/TC1-2008/0118
97424 [242,431], XCU/TC1-2008/0119 [242,431], and XCU/TC1-2008/0120 [325,430] are applied.

97425 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0117 [964], XCU/TC2-2008/0118 [970],
97426 and XCU/TC2-2008/0119 [964] are applied.

97427 **NAME**

97428 mailx — process messages

97429 **SYNOPSIS**97430 **Send Mode**97431 mailx [-s *subject*] *address...*97432 **Receive Mode**

97433 UP mailx -e

97434 mailx [-HiNn] [-F] [-u *user*]97435 mailx -f [-HiNn] [-F] [*file*]97436 **DESCRIPTION**

97437 The *mailx* utility provides a message sending and receiving facility. It has two major modes,
 97438 selected by the options used: Send Mode and Receive Mode.

97439 On systems that do not support the User Portability Utilities option, an application using *mailx*
 97440 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first
 97441 character of one or more lines is <tilde> ('~'), all characters in the input message shall appear in
 97442 the delivered message, but additional characters may be inserted in the message before it is
 97443 retrieved.

97444 UP On systems supporting the User Portability Utilities option, mail-receiving capabilities and other
 97445 interactive features, Receive Mode, described below, also shall be enabled.

97446 **Send Mode**

97447 Send Mode can be used by applications or users to send messages from the text in standard
 97448 input.

97449 UP **Receive Mode**

97450 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this
 97451 interactive mode.

97452 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to
 97453 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the
 97454 message as it is entered.

97455 Incoming mail shall be stored in one or more unspecified locations for each user, collectively
 97456 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system
 97457 mailbox shall be the default place to find new mail. As messages are read, they shall be marked
 97458 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is
 97459 called the **mbox** and is normally located in the directory referred to by the *HOME* environment
 97460 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).
 97461 Messages shall remain in this file until explicitly removed. When the *-f* option is used to read
 97462 mail messages from secondary files, messages shall be retained in those files unless specifically
 97463 removed. All three of these locations—system mailbox, **mbox**, and secondary file ~~file~~ referred
 97464 to in this section as simply “mailboxes”, unless more specific identification is required.

97465 **OPTIONS**

97466 The *mailx* utility shall conform to XBD [Section 12.2](#) (on page 216).

97467 The following options shall be supported. (Only the `-s subject` option shall be required on all
97468 systems. The other options are required only on systems supporting the User Portability Utilities
97469 option.)

97470 UP `-e` Test for the presence of mail in the system mailbox. The *mailx* utility shall write
97471 nothing and exit with a successful return code if there is mail to read.

97472 UP `-f file` Read messages from the file named by the *file* operand instead of the system
97473 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox**
97474 instead of the system mailbox.

97475 UP `-F` Record the message in a file named after the first recipient. The name is the login-
97476 name portion of the address found first on the **To:** line in the mail header.
97477 Overrides the **record** variable, if set (see [Internal Variables in mailx](#), on page 2950).

97478 UP `-H` Write a header summary only.

97479 UP `-i` Ignore interrupts. (See also **ignore**.)

97480 UP `-n` Do not initialize from the system default start-up file. See the EXTENDED
97481 DESCRIPTION section.

97482 UP `-N` Do not write an initial header summary.

97483 `-s subject` Set the **Subject** header field to *subject*. All characters in the *subject* string shall
97484 appear in the delivered message. The results are unspecified if *subject* is longer
97485 than {LINE_MAX} – 10 bytes or contains a <newline>.

97486 UP `-u user` Read the system mailbox of the login name *user*. This shall only be successful if
97487 the invoking user has appropriate privileges to read the system mailbox of that
97488 user.

97489 **OPERANDS**

97490 The following operands shall be supported:

97491 *address* Addressee of message. When `-n` is specified and no user start-up files are accessed
97492 (see the EXTENDED DESCRIPTION section), the user or application shall ensure
97493 this is an address to pass to the mail delivery system. Any system or user start-up
97494 files may enable aliases (see **alias** under [Commands in mailx](#), on page 2953) that
97495 may modify the form of *address* before it is passed to the mail delivery system.

97496 UP *file* A pathname of a file to be read instead of the system mailbox when `-f` is specified.
97497 The meaning of the *file* option-argument shall be affected by the contents of the
97498 **folder** internal variable; see [Internal Variables in mailx](#) (on page 2950).

97499 **STDIN**

97500 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the
97501 UP message to be delivered to the specified addresses. When in Receive Mode, user commands
97502 shall be accepted from *stdin*. If the User Portability Utilities option is not supported, standard
97503 input lines beginning with a <tilde> ('~') character produce unspecified results.

97504 UP If the User Portability Utilities option is supported, then in both Send and Receive Modes,
97505 standard input lines beginning with the escape character (usually <tilde> ('~')) shall affect
97506 processing as described in [Command Escapes in mailx](#) (on page 2962).

97507 **INPUT FILES**

97508 When *mailx* is used as described by this volume of POSIX.1-2017, the *file* option-argument (see
 97509 the **-f** option) and the **mbox** shall be text files containing mail messages, formatted as described
 97510 in the OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a
 97511 file.

97512 **ENVIRONMENT VARIABLES**

97513 UP Some of the functionality described in this section shall be provided on implementations that
 97514 support the User Portability Utilities option as described in the text, and is not further shaded
 97515 for this option.

97516 The following environment variables shall affect the execution of *mailx*:

97517 **DEAD** Determine the pathname of the file in which to save partial messages in case of
 97518 interrupts or delivery errors. The default shall be **dead.letter** in the directory
 97519 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is
 97520 unspecified if the User Portability Utilities option is not supported and *DEAD* is
 97521 not defined with the value **/dev/null**.

97522 **EDITOR** Determine the name of a utility to invoke when the **edit** (see [Commands in mailx](#),
 97523 on page 2953) or **~e** (see [Command Escapes in mailx](#), on page 2962) command is
 97524 XSI used. The default editor is unspecified. [On XSI-conformant systems it is *ed*.](#) The
 97525 effects of this variable are unspecified if the User Portability Utilities option is not
 97526 supported.

97527 **HOME** Determine the pathname of the user's home directory.

97528 **LANG** Provide a default value for the internationalization variables that are unset or null.
 97529 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization
 97530 variables used to determine the values of locale categories.)

97531 **LC_ALL** If set to a non-empty string value, override the values of all the other
 97532 internationalization variables.

97533 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 97534 characters (for example, single-byte as opposed to multi-byte characters in
 97535 arguments and input files) and the handling of case-insensitive address and
 97536 header-field comparisons.

97537 **LC_TIME** This variable may determine the format and contents of the date and time strings
 97538 written by *mailx*. This volume of POSIX.1-2017 specifies the effects of this variable
 97539 only for systems supporting the User Portability Utilities option.

97540 **LC_MESSAGES**

97541 Determine the locale that should be used to affect the format and contents of
 97542 diagnostic messages written to standard error and informative messages written to
 97543 standard output.

97544 **LISTER** Determine a string representing the command for writing the contents of the
 97545 **folder** directory to standard output when the **folders** command is given (see
 97546 **folders** in [Commands in mailx](#), on page 2953). Any string acceptable as a
 97547 *command_string* operand to the *sh -c* command shall be valid. If this variable is null
 97548 or not set, the output command shall be *ls*. The effects of this variable are
 97549 unspecified if the User Portability Utilities option is not supported.

97550 **MAILRC** Determine the pathname of the user start-up file. The default shall be **.mailrc** in the
 97551 directory referred to by the *HOME* environment variable. The behavior of *mailx* is
 97552 unspecified if the User Portability Utilities option is not supported and *MAILRC* is

97553		not defined with the value <code>/dev/null</code> .
97554	<i>MBOX</i>	Determine a pathname of the file to save messages from the system mailbox that have been read. The <code>exit</code> command shall override this function, as shall saving the message explicitly in another file. The default shall be <code>mbox</code> in the directory named by the <code>HOME</code> variable. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
97555		
97556		
97557		
97558		
97559	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <code>LC_MESSAGES</code> .
97560	<i>PAGER</i>	Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a <i>command_string</i> operand to the <code>sh -c</code> command shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the <i>mailx</i> internal variable <code>crf</code> is set to a value less the number of lines in the message; see Internal Variables in mailx (on page 2950). If the <i>PAGER</i> variable is null or not set, the paginator shall be either <i>more</i> or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
97561		
97562		
97563		
97564		
97565		
97566		
97567		
97568		
97569	<i>SHELL</i>	Determine the name of a preferred command interpreter. The default shall be <i>sh</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.
97570		
97571		
97572	<i>TERM</i>	If the internal variable <code>screen</code> is not specified, determine the name of the terminal type to indicate in an unspecified manner the number of lines in a screenful of headers. If <i>TERM</i> is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
97573		
97574		
97575		
97576		
97577	<i>TZ</i>	This variable may determine the timezone used to calculate date and time strings written by <i>mailx</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
97578		
97579		
97580	<i>VISUAL</i>	Determine a pathname of a utility to invoke when the <code>visual</code> command (see Commands in mailx , on page 2953) or <code>~v</code> command-escape (see Command Escapes in mailx , on page 2962) is used. If this variable is null or not set, the full-screen editor shall be <i>vi</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.
97581		
97582		
97583		
97584		
97585	ASYNCHRONOUS EVENTS	
97586		When <i>mailx</i> is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals.
97587		
97588	UP	In <code>Receive Mode</code> , or in <code>Send Mode</code> when standard input is a terminal, if a <code>SIGINT</code> signal is received:
97589		
97590	UP	1. If in command mode, the current command, if there is one, shall be aborted, and a command-mode prompt shall be written.
97591		
97592		2. If in input mode:
97593	UP	a. If <code>ignore</code> is set, <i>mailx</i> shall write " <code>@\n</code> ", discard the current input line, and continue processing, bypassing the message-abort mechanism described in item 2b.
97594		
97595		

- 97596 UP b. If the interrupt was received while sending mail, either when in Receive Mode or
 97597 **in** Send Mode, a message shall be written, and another subsequent interrupt, with
 97598 no other intervening characters typed, shall be required to abort the mail message.
 97599 UP If in Receive Mode and another interrupt is received, a command-mode prompt
 97600 shall be written. If in Send Mode and another interrupt is received, *mailx* shall
 97601 terminate with a non-zero status.

97602 In both cases listed in item b, if the message is not empty:

- 97603 UP i. If **save** is enabled and the file named by *DEAD* can be created, the message
 97604 shall be written to the file named by *DEAD*. If the file exists, the message
 97605 shall be written to replace the contents of the file.
- 97606 UP ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the
 97607 message shall not be saved.

97608 The *mailx* utility shall take the standard action for all other signals.

97609 **STDOUT**

97610 In command and input modes, all output, including prompts and messages, shall be written to
 97611 standard output.

97612 **STDERR**

97613 The standard error shall be used only for diagnostic messages.

97614 **OUTPUT FILES**

97615 Various *mailx* commands and command escapes can create or add to files, including the **mbox**,
 97616 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of
 97617 POSIX.1-2017, these files shall be text files, formatted as follows:

97618 line beginning with **From**<space>
 97619 [one or more *header-lines*; see [Commands in mailx](#) (on page 2953)]
 97620 *empty line*
 97621 [zero or more *body lines*
 97622 *empty line*]
 97623 [line beginning with **From**<space>...]

97624 where each message begins with the **From** <space> line shown, preceded by the beginning of
 97625 the file or an empty line. (The **From** <space> line is considered to be part of the message header,
 97626 but not one of the header-lines referred to in [Commands in mailx](#) (on page 2953); thus, it shall
 97627 not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the
 97628 **From** <space> line and any additional header lines are unspecified, except that none shall be
 97629 empty. The format of a message body line is also unspecified, except that no line following an
 97630 empty line shall start with **From** <space>; *mailx* shall modify any such user-entered message
 97631 body lines (following an empty line and beginning with **From** <space>) by adding one or more
 97632 characters to precede the 'F'; it may add these characters to **From** <space> lines that are not
 97633 preceded by an empty line.

97634 When a message from the system mailbox or entered by the user is not a text file, it is
 97635 implementation-defined how such a message is stored in files written by *mailx*.

97636 **EXTENDED DESCRIPTION**

97637 UP The functionality in the entire EXTENDED DESCRIPTION section shall be provided on
 97638 implementations supporting the User Portability Utilities option. The functionality described in
 97639 this section shall be provided on implementations that support the User Portability Utilities
 97640 option (and the rest of this section is not further shaded for this option).

97641 The *mailx* utility need not support for all character encodings in all circumstances. For example,
 97642 inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not
 97643 be portable to non-internationalized systems, and so on. Under these circumstances, it is
 97644 recommended that only characters defined in the ISO/IEC 646:1991 standard International
 97645 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

97646 When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of
 97647 header-summary lines (if `-N` was not specified and there are messages, see below), followed by
 97648 a prompt indicating that *mailx* can accept regular commands (see [Commands in mailx](#), on page
 97649 2953); this is termed *command mode*. The page of header-summary lines shall contain the first
 97650 new message if there are new messages, or the first unread message if there are unread
 97651 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and
 97652 standard input is a terminal, if no subject is specified on the command line and the **asksub**
 97653 variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input
 97654 mode. This input mode shall also be entered when using one of the Receive Mode synopsis
 97655 forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or
 97656 **mail** commands and standard input is a terminal. When the message is typed and the end of the
 97657 message is encountered, the message shall be passed to the mail delivery software. Commands
 97658 can be entered by beginning a line with the escape character (by default, <tilde> (' ~ ')) followed
 97659 by a single command letter and optional arguments. See [Commands in mailx](#) (on page 2953) for
 97660 a summary of these commands. It is unspecified what effect these commands will have if
 97661 standard input is not a terminal when a message is entered using either the Send Mode
 97662 synopsis, or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

97663 **Note:** For notational convenience, this section uses the default escape character, <tilde>, in all
 97664 references and examples.

97665 At any time, the behavior of *mailx* shall be governed by a set of environmental and internal
 97666 variables. These are flags and valued parameters that can be set and cleared via the *mailx* **set**
 97667 and **unset** commands.

97668 Regular commands are of the form:

```
97669 [command] [msglist] [argument ...]
```

97670 If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands
 97671 shall be recognized by the escape character, and lines not treated as commands shall be taken as
 97672 input for the message.

97673 In command mode, each message shall be assigned a sequential number, starting with 1.

97674 All messages have a state that shall affect how they are displayed in the header summary and
 97675 how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a
 97676 *current* message, which shall be marked by a '>' at the beginning of a line in the header
 97677 summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current
 97678 message shall be the first new message, if there is a new message, or the first unread message if
 97679 there is an unread message, or the first message if there are any messages, or unspecified if there
 97680 are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*)
 97681 or an optional single message (*message*) on which to operate shall leave the current message set
 97682 to the highest-numbered message of the messages specified, unless the command deletes
 97683 messages, in which case the current message shall be set to the first undeleted message (that is, a
 97684 message not in the deleted state) after the highest-numbered message deleted by the command,
 97685 if one exists, or the first undeleted message before the highest-numbered message deleted by the
 97686 command, if one exists, or to an unspecified value if there are no remaining undeleted messages.
 97687 All messages shall be in one of the following states:

97688	<i>new</i>	The message is present in the system mailbox and has not been viewed by the user or moved to any other state. Messages in state <i>new</i> when <i>mailx</i> quits shall be retained in the system mailbox.
97689		
97690		
97691	<i>unread</i>	The message has been present in the system mailbox for more than one invocation of <i>mailx</i> and has not been viewed by the user or moved to any other state. Messages in state <i>unread</i> when <i>mailx</i> quits shall be retained in the system mailbox.
97692		
97693		
97694	<i>read</i>	The message has been processed by one of the following commands: ~f, ~m, ~F, ~M, copy, mbox, next, pipe, print, Print, top, type, Type, undelete . The delete, dp, and dt commands may also cause the next message to be marked as <i>read</i> , depending on the value of the autoprint variable. Messages that are in the system mailbox and in state <i>read</i> when <i>mailx</i> quits shall be saved in the mbox , unless the internal variable hold was set. Messages that are in the mbox or in a secondary mailbox and in state <i>read</i> when <i>mailx</i> quits shall be retained in their current location.
97695		
97696		
97697		
97698		
97699		
97700		
97701		
97702	<i>deleted</i>	The message has been processed by one of the following commands: delete, dp, dt . Messages in state <i>deleted</i> when <i>mailx</i> quits shall be deleted. Deleted messages shall be ignored until <i>mailx</i> quits or changes mailboxes or they are specified to the undelete command; for example, the message specification <i>/string</i> shall only search the subject lines of messages that have not yet been deleted, unless the command operating on the list of messages is undelete . No deleted message or deleted message header shall be displayed by any <i>mailx</i> command other than undelete .
97703		
97704		
97705		
97706		
97707		
97708		
97709		
97710	<i>preserved</i>	The message has been processed by a preserve command. When <i>mailx</i> quits, the message shall be retained in its current location.
97711		
97712	<i>saved</i>	The message has been processed by one of the following commands: save or write . If the current mailbox is the system mailbox, and the internal variable keepsave is set, messages in the state <i>saved</i> shall be saved to the file designated by the <i>MBOX</i> variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is the system mailbox, messages in the state <i>saved</i> shall be deleted from the current mailbox, when the quit or file command is used to exit the current mailbox.
97713		
97714		
97715		
97716		
97717		
97718		The header-summary line for each message shall indicate the state of the message.
97719		Many commands take an optional list of messages (<i>msglist</i>) on which to operate, which defaults to the current message. A <i>msglist</i> is a list of message specifications separated by <blank> characters, which can include:
97720		
97721		
97722	<i>n</i>	Message number <i>n</i> .
97723	+	The next undeleted message, or the next deleted message for the undelete command.
97724	-	The next previous undeleted message, or the next previous deleted message for the undelete command.
97725		
97726	.	The current message.
97727	^	The first undeleted message, or the first deleted message for the undelete command.
97728	\$	The last message.
97729	*	All messages.

- 97730 n-m An inclusive range of message numbers.
- 97731 *address* All messages from *address*; any address as shown in a header summary shall be
97732 matchable in this form.
- 97733 */string* All messages with *string* in the subject line (case ignored).
- 97734 :c All messages of type *c*, where *c* shall be one of:
- 97735 d Deleted messages.
- 97736 n New messages.
- 97737 o Old messages (any not in state *read* or *new*).
- 97738 r Read messages.
- 97739 u Unread messages.

97740 Other commands take an optional message (*message*) on which to operate, which defaults to the
97741 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more
97742 than one message is specified, only the first shall be operated on.

97743 Other arguments are usually arbitrary strings whose usage depends on the command involved.

97744 **Start-Up in mailx**

97745 At start-up time, *mailx* shall take the following steps in sequence:

- 97746 1. Establish all variables at their stated default values.
- 97747 2. Process command line options, overriding corresponding default values.
- 97748 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables
97749 that are present in the environment, overriding the corresponding default values.
- 97750 4. Read *mailx* commands from an unspecified system start-up file, unless the **-n** option is
97751 given, to initialize any internal *mailx* variables and aliases.
- 97752 5. Process the user start-up file of *mailx* commands named in the user *MAILRC* variable.

97753 Most regular *mailx* commands are valid inside start-up files, the most common use being to set
97754 up initial display options and alias lists. The following commands shall be invalid in a start-up
97755 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**.
97756 Any errors in a start-up file shall either cause *mailx* to terminate with a diagnostic message and a
97757 non-zero status or to continue after writing a diagnostic message, ignoring the remainder of the
97758 lines in the file.

97759 A blank line in a start-up file shall be ignored.

97760 **Internal Variables in mailx**

97761 The following variables are internal *mailx* variables. Each internal variable can be set via the
97762 *mailx set* command at any time. The **unset** and **set no name** commands can be used to erase
97763 variables.

97764 In the following list, variables shown as:

97765 `variable`

97766 represent Boolean values. Variables shown as:

97767 `variable=value`

- 97768 shall be assigned string or numeric values. For string values, the rules in [Commands in mailx](#)
 97769 (on page 2953) concerning filenames and quoting shall also apply.
- 97770 The defaults specified here may be changed by the unspecified system start-up file unless the
 97771 user specifies the **-n** option.
- 97772 **allnet** All network names whose login name components match shall be treated as
 97773 identical. This shall cause the *msglist* message specifications to behave similarly.
 97774 The default shall be **noallnet**. See also the **alternates** command and the **metoo**
 97775 variable.
- 97776 **append** Append messages to the end of the **mbox** file upon termination instead of placing
 97777 them at the beginning. The default shall be **noappend**. This variable shall not
 97778 affect the **save** command when saving to **mbox**.
- 97779 **ask, asksub** Prompt for a subject line on outgoing mail if one is not specified on the command
 97780 line with the **-s** option. The **ask** and **asksub** forms are synonyms; the system shall
 97781 refer to **asksub** and **noasksub** in its messages, but shall accept **ask** and **noask** as
 97782 user input to mean **asksub** and **noasksub**. It shall not be possible to set both **ask**
 97783 and **noasksub**, or **noask** and **asksub**. The default shall be **asksub**, but no
 97784 prompting shall be done if standard input is not a terminal.
- 97785 **askbcc** Prompt for the blind copy list. The default shall be **noaskbcc**.
- 97786 **askcc** Prompt for the copy list. The default shall be **noaskcc**.
- 97787 **autoprint** Enable automatic writing of messages after **delete** and **undelete** commands. The
 97788 default shall be **noautoprint**.
- 97789 **bang** Enable the special-case treatment of <exclamation-mark> characters ('!') in
 97790 escape command lines; see the **escape** command and [Command Escapes in mailx](#)
 97791 (on page 2962). The default shall be **nobang**, disabling the expansion of '!' in the
 97792 *command* argument to the **!** command and the **~<command** escape.
- 97793 **cmd=command**
 97794 Set the default command to be invoked by the **pipe** command. The default shall be
 97795 **nocmd**.
- 97796 **crt=number** Pipe messages having more than *number* lines through the command specified by
 97797 the value of the *PAGER* variable. The default shall be **nocrt**. If it is set to null, the
 97798 value used is implementation-defined.
- 97799 XSI **debug** Enable verbose diagnostics for debugging. Messages are not delivered. The
 97800 default shall be **nodebug**.
- 97801 **dot** When **dot** is set, a <period> on a line by itself during message input from a
 97802 terminal shall also signify end-of-file (in addition to normal end-of-file). The
 97803 default shall be **nodot**. If **ignoreeof** is set (see below), a setting of **nodot** shall be
 97804 ignored and the <period> is the only method to terminate input mode.
- 97805 **escape=c** Set the command escape character to be the character 'c'. By default, the
 97806 command escape character shall be <tilde>. If **escape** is unset, <tilde> shall be
 97807 used; if it is set to null, command escaping shall be disabled.
- 97808 **flipr** Reverse the meanings of the **R** and **r** commands. The default shall be **noflipr**.
- 97809 **folder=directory**
 97810 The default directory for saving mail files. User-specified filenames beginning with
 97811 a <plus-sign> ('+') shall be expanded by preceding the filename with this

97812		directory name to obtain the real pathname. If <i>directory</i> does not start with a
97813		<slash> ('/'), the contents of <i>HOME</i> shall be prefixed to it. The default shall be
97814		nofolder . If folder is unset or set to null, user-specified filenames beginning with
97815		'+' shall refer to files in the current directory that begin with the literal '+'
97816		character. See also outfolder below. The folder value need not affect the processing
97817		of the files named in <i>MBOX</i> and <i>DEAD</i> .
97818	header	Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The
97819		default shall be header .
97820	hold	Preserve all messages that are read in the system mailbox instead of putting them
97821		in the mbox save file. The default shall be nohold .
97822	ignore	Ignore interrupts while entering messages. The default shall be noignore .
97823	ignoreeof	Ignore normal end-of-file during message input. Input can be terminated only by
97824		entering a <period> ('.') on a line by itself or by the ~. command escape. The
97825		default shall be noignoreeof . See also dot above.
97826	indentprefix=string	
97827		A string that shall be added as a prefix to each line that is inserted into the message
97828		by the ~m command escape. This variable shall default to one <tab>.
97829	keep	When a system mailbox, secondary mailbox, or mbox is empty, truncate it to zero
97830		length instead of removing it. The default shall be nokeep .
97831	keepsave	Keep the messages that have been saved from the system mailbox into other files
97832		in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default
97833		shall be nokeepsave .
97834	metoo	Suppress the deletion of the login name of the user from the recipient list when
97835		replying to a message or sending to a group. The default shall be nometoo .
97836	XSI onehop	When responding to a message that was originally sent to several recipients, the
97837		other recipient addresses are normally forced to be relative to the originating
97838		author's machine for the response. This flag disables alteration of the recipients'
97839		addresses, improving efficiency in a network where all machines can send directly
97840		to all other machines (that is, one hop away). The default shall be noonehop .
97841	outfolder	Cause the files used to record outgoing messages to be located in the directory
97842		specified by the folder variable unless the pathname is absolute. The default shall
97843		be nooutfolder . See the record variable.
97844	page	Insert a <form-feed> after each message sent through the pipe created by the pipe
97845		command. The default shall be nopage .
97846	prompt=string	
97847		Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if noprompt is set, no
97848		prompting shall occur. The default shall be to prompt with the string "? ".
97849	quiet	Refrain from writing the opening message and version when entering <i>mailx</i> . The
97850		default shall be noquiet .
97851	record=file	Record all outgoing mail in the file with the pathname <i>file</i> . The default shall be
97852		norecord . See also outfolder above.
97853	save	Enable saving of messages in the dead-letter file on interrupt or delivery error. See
97854		the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be save .

- 97855 **screen=number**
 97856 Set the number of lines in a screenful of headers for the **headers** and **z** commands.
 97857 If **screen** is not specified, a value based on the terminal type identified by the
 97858 *TERM* environment variable, the window size, the baud rate, or some combination
 97859 of these shall be used.
- 97860 **sendwait** Wait for the background mailer to finish before returning. The default shall be
 97861 **nosendwait**.
- 97862 **showto** When the sender of the message was the user who is invoking *mailx*, write the
 97863 information from the **To:** line instead of the **From:** line in the header summary. The
 97864 default shall be **noshowto**.
- 97865 **sign=string** Set the variable inserted into the text of a message when the **~a** command escape is
 97866 given. The default shall be **nosign**. The character sequences '\t' and '\n' shall
 97867 be recognized in the variable as <tab> and <newline> characters, respectively. (See
 97868 also **~i** in [Command Escapes in mailx](#) (on page 2962).)
- 97869 **Sign=string** Set the variable inserted into the text of a message when the **~A** command escape is
 97870 given. The default shall be **noSign**. The character sequences '\t' and '\n' shall
 97871 be recognized in the variable as <tab> and <newline> characters, respectively.
- 97872 **toplines=number**
 97873 Set the number of lines of the message to write with the **top** command. The default
 97874 shall be 5.

97875 **Commands in mailx**

97876 The following *mailx* commands shall be provided. In the following list, header refers to lines
 97877 from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines
 97878 within the header that begin with one or more non-white-space characters, immediately
 97879 followed by a <colon> and white space and continuing until the next line beginning with a non-
 97880 white-space character or an empty line. Header-field refers to the portion of a header line prior
 97881 to the first <colon> in that line.

97882 For each of the commands listed below, the command can be entered as the abbreviation (those
 97883 characters in the Synopsis command word preceding the '['), the full command (all characters
 97884 shown for the command word, omitting the '[' and ']'), or any truncation of the full
 97885 command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the
 97886 Synopsis) can be entered as **ex**, **exi**, or **exit**.

97887 The arguments to commands can be quoted, using the following methods:

97888 An argument can be enclosed between paired double-quotes (" ") or single-quotes (' ');
 97889 any white space, shell word expansion, or <backslash> characters within the quotes shall
 97890 be treated literally as part of the argument. A double-quote shall be treated literally within
 97891 single-quotes and *vice versa*. These special properties of the <quotation-mark> characters
 97892 shall occur only when they are paired at the beginning and end of the argument.

97893 A <backslash> outside of the enclosing quotes shall be discarded and the following
 97894 character treated literally as part of the argument.

97895 An unquoted <backslash> at the end of a command line shall be discarded and the next
 97896 line shall continue the command.

97897 Filenames, where expected, shall be subjected to the following transformations, in sequence:

97898 If the filename begins with an unquoted <plus-sign>, and the **folder** variable is defined
 97899 (see the **folder** variable), the <plus-sign> shall be replaced by the value of the **folder**
 97900 variable followed by a <slash>. If the **folder** variable is unset or is set to null, the filename
 97901 shall be unchanged.

97902 Shell word expansions shall be applied to the filename (see [Section 2.6](#), on page 2353). If
 97903 more than a single pathname results from this expansion and the command is expecting
 97904 one file, the effects are unspecified.

97905 **Declare Aliases**

97906 *Synopsis:* a[lias] [*alias* [*address...*]]
 97907 g[roup] [*alias* [*address...*]]

97908 Add the given addresses to the alias specified by *alias*. The names shall be substituted when
 97909 *alias* is used as a recipient address specified by the user in an outgoing message (that is, other
 97910 recipients addressed indirectly through the **reply** command shall not be substituted in this
 97911 manner). Mail address alias substitution shall apply only when the alias string is used as a full
 97912 address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If
 97913 no arguments are given, write a listing of the current aliases to standard output. If only an *alias*
 97914 argument is given, write a listing of the specified alias to standard output. These listings need
 97915 not reflect the same order of addresses that were entered.

97916 **Declare Alternatives**

97917 *Synopsis:* alt[ernates] *name...*

97918 (See also the **metoo** variable.) Declare a list of alternative names for the user's login. When
 97919 responding to a message, these names shall be removed from the list of recipients for the
 97920 response. The comparison of names shall be in a case-insensitive manner. With no arguments,
 97921 **alternates** shall write the current list of alternative names.

97922 **Change Current Directory**

97923 *Synopsis:* cd [*directory*]
 97924 ch[dir] [*directory*]

97925 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

97926 **Copy Messages**

97927 *Synopsis:* c[opy] [*file*]
 97928 c[opy] [*msglist*] *file*
 97929 C[opy] [*msglist*]

97930 Copy messages to the file named by the pathname *file* without marking the messages as saved.
 97931 Otherwise, it shall be equivalent to the **save** command.

97932 In the capitalized form, save the specified messages in a file whose name is derived from the
 97933 author of the message to be saved, without marking the messages as saved. Otherwise, it shall
 97934 be equivalent to the **Save** command.

97935 **Delete Messages**97936 *Synopsis:* d[delete] [msglist]

97937 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see
 97938 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there
 97939 are messages remaining after the **delete** command, the current message shall be written as
 97940 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be
 97941 written.

97942 **Discard Header Fields**97943 *Synopsis:* di[scard] [header-field...]

97944 ig[nore] [header-field...]

97945 Suppress the specified header fields when writing messages. Specified *header-fields* shall be
 97946 added to the list of suppressed header fields. Examples of header fields to ignore are **status** and
 97947 **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall
 97948 override this command. The comparison of header fields shall be in a case-insensitive manner. If
 97949 no arguments are specified, write a list of the currently suppressed header fields to standard
 97950 output; the listing need not reflect the same order of header fields that were entered.

97951 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.97952 **Delete Messages and Display**97953 *Synopsis:* dp [msglist]

97954 dt [msglist]

97955 Delete the specified messages as described for the **delete** command, except that the **autoprint**
 97956 variable shall have no effect, and the current message shall be written only if it was set to a
 97957 message after the last message deleted by the command. Otherwise, an informational message
 97958 to the effect that there are no further messages in the mailbox shall be written, followed by the
 97959 *mailx* prompt.

97960 **Echo a String**97961 *Synopsis:* ec[ho] string ...97962 Echo the given strings, equivalent to the shell *echo* utility.97963 **Edit Messages**97964 *Synopsis:* e[dit] [msglist]

97965 Edit the given messages. The messages shall be placed in a temporary file and the utility named
 97966 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is
 97967 unspecified.

97968 The **edit** command does not modify the contents of those messages in the mailbox.

97969 **Exit**

97970 *Synopsis:* ex[it]
 97971 x[it]

97972 Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also
 97973 **quit**).

97974 **Change Folder**

97975 *Synopsis:* fi[le] [*file*]
 97976 fold[er] [*file*]

97977 Quit (see the **quit** command) from the current file of messages and read in the file named by the
 97978 pathname *file*. If no argument is given, the name and status of the current mailbox shall be
 97979 written.

97980 Several unquoted special characters shall be recognized when used as *file* names, with the
 97981 following substitutions:

97982 % The system mailbox for the invoking user.

97983 %*user* The system mailbox for *user*.

97984 # The previous file.

97985 & The current **mbox**.

97986 +*file* The named file in the **folder** directory. (See the **folder** variable.)

97987 The default file shall be the current mailbox.

97988 **Display List of Folders**

97989 *Synopsis:* folders

97990 Write the names of the files in the directory set by the **folder** variable. The command specified
 97991 by the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES
 97992 section).

97993 **Follow Up Specified Messages**

97994 *Synopsis:* fo[llowup] [*message*]
 97995 F[ollowup] [*msglist*]

97996 In the lowercase form, respond to a message, recording the response in a file whose name is
 97997 derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.

97998 In the capitalized form, respond to the first message in the *msglist*, sending the message to the
 97999 author of each message in the *msglist*. The subject line shall be taken from the first message and
 98000 the response shall be recorded in a file whose name is derived from the author of the first
 98001 message. See also the **Save** and **Copy** commands and **outfolder**.

98002 Both forms shall override the **record** variable, if set.

98003 Display Header Summary for Specified Messages

98004 *Synopsis:* f[rom] [msglist]

98005 Write the header summary for the specified messages.

98006 Display Header Summary

98007 *Synopsis:* h[eaders] [message]

98008 Write the page of headers that includes the message specified. If the *message* argument is not
98009 specified, the current message shall not change. However, if the *message* argument is specified,
98010 the current message shall become the message that appears at the top of the page of headers that
98011 includes the message specified. The **screen** variable sets the number of headers per page. See
98012 also the **z** command.

98013 Help

98014 *Synopsis:* hel[p]

98015 ?

98016 Write a summary of commands.

98017 Hold Messages

98018 *Synopsis:* ho[ld] [msglist]

98019 pre[serve] [msglist]

98020 Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall
98021 override any commands that might previously have marked the messages to be deleted. During
98022 the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve*
98023 marking of a message.

98024 Execute Commands Conditionally

98025 *Synopsis:* i[f] s|r

98026 *mail-commands*

98027 el[se]

98028 *mail-commands*

98029 en[dif]

98030 Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an
98031 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be
98032 executed only in Receive Mode.

98033 List Available Commands

98034 *Synopsis:* l[ist]

98035 Write a list of all commands available. No explanation shall be given.

98036 **Mail a Message**98037 *Synopsis:* m[ail] address...

98038 Mail a message to the specified addresses or aliases.

98039 **Direct Messages to mbox**98040 *Synopsis:* mb[ox] [msglist]98041 Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally.
98042 See *MBOX*. See also the **exit** and **quit** commands.98043 **Process Next Specified Message**98044 *Synopsis:* n[ext] [message]98045 If the current message has not been written (for example, by the **print** command) since *mailx*
98046 started or since any other message was the current message, behave as if the **print** command
98047 was entered. Otherwise, if there is an undeleted message after the current message, make it the
98048 current message and behave as if the **print** command was entered. Otherwise, an informational
98049 message to the effect that there are no further messages in the mailbox shall be written, followed
98050 by the *mailx* prompt. Should the current message location be the result of an immediately
98051 preceding **hold**, **mbox**, **preserve**, or **touch** command, **next** will act as if the current message has
98052 already been written.98053 **Pipe Message**98054 *Synopsis:* pi[pe] [[msglist] command]
98055 | [[msglist] command]98056 Pipe the messages through the given *command* by invoking the command interpreter specified
98057 by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure
98058 that the command is given as a single argument. Quoting, described previously, can be used to
98059 accomplish this. If no arguments are given, the current message shall be piped through the
98060 command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed>
98061 shall be inserted after each message.98062 **Display Message with Headers**98063 *Synopsis:* P[rint] [msglist]
98064 T[ype] [msglist]98065 Write the specified messages, including all header lines, to standard output. Override
98066 suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages
98067 longer than the number of lines specified by the **crt** variable shall be paged through the
98068 command specified by the *PAGER* environment variable.98069 **Display Message**98070 *Synopsis:* p[rint] [msglist]
98071 t[ype] [msglist]98072 Write the specified messages to standard output. If **crt** is set, the messages longer than the
98073 number of lines specified by the **crt** variable shall be paged through the command specified by
98074 the *PAGER* environment variable.

98075 **Quit**

98076 *Synopsis:* q[uit]
 98077 end-of-file

98078 Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system
 98079 mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless
 98080 **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages
 98081 in the mailbox.

98082 **Reply to a Message List**

98083 *Synopsis:* R[e]ply [msglist]
 98084 R[es]pond [msglist]

98085 Mail a reply message to the sender of each message in the *msglist*. The subject line shall be
 98086 formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject
 98087 from the first message. If **record** is set to a filename, the response shall be saved at the end of that
 98088 file.

98089 See also the **flipr** variable.

98090 **Reply to a Message**

98091 *Synopsis:* r[e]ply [message]
 98092 r[es]pond [message]

98093 Mail a reply message to all recipients included in the header of the message. The subject line
 98094 shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the
 98095 subject from the message. If **record** is set to a filename, the response shall be saved at the end of
 98096 that file.

98097 See also the **flipr** variable.

98098 **Retain Header Fields**

98099 *Synopsis:* ret[ain] [header-field...]

98100 Retain the specified header fields when writing messages. This command shall override all
 98101 **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive
 98102 manner. If no arguments are specified, write a list of the currently retained header fields to
 98103 standard output; the listing need not reflect the same order of header fields that were entered.

98104 **Save Messages**

98105 *Synopsis:* s[ave] [file]
 98106 s[ave] [msglist] file
 98107 S[ave] [msglist]

98108 Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file*
 98109 argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be
 98110 appended to the file. The message shall be put in the state *saved*, and shall behave as specified in
 98111 the description of the *saved* state when the current mailbox is exited by the **quit** or **file**
 98112 command.

98113 In the capitalized form, save the specified messages in a file whose name is derived from the
 98114 author of the first message. The name of the file shall be taken to be the author's name with all
 98115 network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and

98116 **outfolder** variable.

98117 **Set Variables**

98118 *Synopsis:* `se[t] [name]=[string] ...] [name=number ...] [noname ...]`

98119 Define one or more variables called *name*. The variable can be given a null, string, or numeric
98120 value. Quoting and <backslash>-escapes can occur anywhere in *string*, as described previously,
98121 as if the *string* portion of the argument were the entire argument. The forms *name* and *name=*
98122 shall be equivalent to *name=""* for variables that take string values. The **set** command without
98123 arguments shall write a list of all defined variables and their values. The **no name** form shall be
98124 equivalent to **unset name**.

98125 **Invoke a Shell**

98126 *Synopsis:* `sh[ell]`

98127 Invoke an interactive command interpreter (see also *SHELL*).

98128 **Display Message Size**

98129 *Synopsis:* `si[ze] [msglist]`

98130 Write the size in bytes of each of the specified messages.

98131 **Read mailx Commands From a File**

98132 *Synopsis:* `so[urce] file`

98133 Read and execute commands from the file named by the pathname *file* and return to command
98134 mode.

98135 **Display Beginning of Messages**

98136 *Synopsis:* `to[p] [msglist]`

98137 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken
98138 as the number of lines to write. The default shall be 5.

98139 **Touch Messages**

98140 *Synopsis:* `tou[ch] [msglist]`

98141 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a
98142 file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

98143 **Delete Aliases**

98144 *Synopsis:* `una[lias] [alias]...`

98145 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

98146 **Undelete Messages**98147 *Synopsis:* `u[ndelete] [msglist]`98148 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last
98149 message of those restored shall be written. If *msglist* is not specified, the message shall be
98150 selected as follows:98151 If there are any deleted messages that follow the current message, the first of these shall be
98152 chosen.

98153 Otherwise, the last deleted message that also precedes the current message shall be chosen.

98154 **Unset Variables**98155 *Synopsis:* `uns[et] name...`

98156 Cause the specified variables to be erased.

98157 **Edit Message with Full-Screen Editor**98158 *Synopsis:* `v[isual] [msglist]`98159 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,
98160 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The
98161 default editor shall be *vi*.98162 The **visual** command does not modify the contents of those messages in the mailbox.98163 **Write Messages to a File**98164 *Synopsis:* `w[rite] [msglist] file`98165 Write the given messages to the file specified by the pathname *file*, minus the message header.
98166 Otherwise, it shall be equivalent to the **save** command.98167 **Scroll Header Display**98168 *Synopsis:* `z[+|-]`98169 Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if
98170 '-' is specified) one screenful. The number of headers written shall be set by the **screen**
98171 variable.98172 **Invoke Shell Command**98173 *Synopsis:* `!command`98174 Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See
98175 also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be
98176 replaced with the command executed by the previous ! command or ~! command escape.

98177 **Null Command**98178 *Synopsis:* # *comment*98179 This null command (comment) shall be ignored by *mailx*.98180 **Display Current Message Number**98181 *Synopsis:* =

98182 Write the current message number.

98183 **Command Escapes in mailx**98184 The following commands can be entered only from input mode, by beginning a line with the
98185 escape character (by default, <tilde> ('~')). See the **escape** variable description for changing
98186 this special character. The format for the commands shall be:

98187 <escape-character><command-char><separator> [<arguments>]

98188 where the <separator> can be zero or more <blank> characters.

98189 In the following descriptions, the application shall ensure that the argument *command* (but not
98190 *mailx-command*) is a shell command string. Any string acceptable to the command interpreter
98191 specified by the *SHELL* variable when it is invoked as *SHELL -c command_string* shall be valid.
98192 The command can be presented as multiple arguments (that is, quoting is not required).98193 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send
98194 Mode and produce unspecified results.98195 ~! *command* Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and
98196 *command*; and then return to input mode. If the **bang** variable is set, each
98197 unescaped occurrence of '!' in *command* shall be replaced with the command
98198 executed by the previous ! command or ~! command escape.

98199 ~. Simulate end-of-file (terminate message input).

98200 ~: *mailx-command*, ~_ *mailx-command*
98201 Perform the command-level request.

98202 ~? Write a summary of command escapes.

98203 ~A This shall be equivalent to ~i **Sign**.98204 ~a This shall be equivalent to ~i **sign**.98205 ~b *name...* Add the *names* to the blind carbon copy (**Bcc**) list.98206 ~c *name...* Add the *names* to the carbon copy (**Cc**) list.98207 ~d Read in the dead-letter file. See *DEAD* for a description of this file.98208 ~e Invoke the editor, as specified by the *EDITOR* environment variable, on the partial
98209 message.98210 ~f [*msglist*] Forward the specified messages. The specified messages shall be inserted into the
98211 current message without alteration. This command escape also shall insert
98212 message headers into the message with field selection affected by the **discard**,
98213 **ignore**, and **retain** commands.

- 98214 **~F** [*msglist*] This shall be the equivalent of the **~f** command escape, except that all headers shall
 98215 be included in the message, regardless of previous **discard**, **ignore**, and **retain**
 98216 commands.
- 98217 **~h** If standard input is a terminal, prompt for a **Subject** line and the **To**, **Cc**, and **Bcc**
 98218 lists. Other implementation-defined headers may also be presented for editing. If
 98219 the field is written with an initial value, it can be edited as if it had just been typed.
- 98220 **~i** *string* Insert the value of the named variable, followed by a <newline>, into the text of
 98221 the message. If the string is unset or null, the message shall not be changed.
- 98222 **~m** [*msglist*] Insert the specified messages into the message, prefixing non-empty lines with the
 98223 string in the **indentprefix** variable. This command escape also shall insert message
 98224 headers into the message, with field selection affected by the **discard**, **ignore**, and
 98225 **retain** commands.
- 98226 **~M** [*msglist*] This shall be the equivalent of the **~m** command escape, except that all headers
 98227 shall be included in the message, regardless of previous **discard**, **ignore**, and **retain**
 98228 commands.
- 98229 **~p** Write the message being entered. If the message is longer than **crt** lines (see
 98230 [Internal Variables in mailx](#), on page 2950), the output shall be paginated as
 98231 described for the **PAGER** variable.
- 98232 **~q** Quit (see the **quit** command) from input mode by simulating an interrupt. If the
 98233 body of the message is not empty, the partial message shall be saved in the dead-
 98234 letter file. See **DEAD** for a description of this file.
- 98235 **~r** *file*, **~<** *file*, **~r** *!command*, **~<** *!command*
 98236 Read in the file specified by the pathname *file*. If the argument begins with an
 98237 <exclamation-mark> ('!'), the rest of the string shall be taken as an arbitrary
 98238 system command; the command interpreter specified by **SHELL** shall be invoked
 98239 with two arguments: **-c** and *command*. The standard output of *command* shall be
 98240 inserted into the message.
- 98241 **~s** *string* Set the subject line to *string*.
- 98242 **~t** *name...* Add the given *names* to the **To** list.
- 98243 **~v** Invoke the full-screen editor, as specified by the **VISUAL** environment variable, on
 98244 the partial message.
- 98245 **~w** *file* Write the partial message, without the header, onto the file named by the
 98246 pathname *file*. The file shall be created or the message shall be appended to it if
 98247 the file exists.
- 98248 **~x** Exit as with **~q**, except the message shall not be saved in the dead-letter file.
- 98249 **~|** *command* Pipe the body of the message through the given *command* by invoking the
 98250 command interpreter specified by **SHELL** with two arguments: **-c** and *command*. If
 98251 the *command* returns a successful exit status, the standard output of the command
 98252 shall replace the message. Otherwise, the message shall remain unchanged. If the
 98253 *command* fails, an error message giving the exit status shall be written.

98254 **EXIT STATUS**98255 UP When the `-e` option is specified, the following exit values are returned:

98256 0 Mail was found.

98257 >0 Mail was not found or an error occurred.

98258 Otherwise, the following exit values are returned:

98259 0 Successful completion; note that this status implies that all messages were *sent*, but it gives
98260 no assurances that any of them were actually *delivered*.

98261 >0 An error occurred.

98262 **CONSEQUENCES OF ERRORS**98263 UP When in `input mode (Receive Mode)` or Send Mode:98264 If an error is encountered processing an input line beginning with a <tilde> ('~')
98265 UP character, (see [Command Escapes in mailx](#), on page 2962), a diagnostic message shall be
98266 written to standard error, and the message being composed may be modified, but this
98267 condition shall not prevent the message from being sent.

98268 Other errors shall prevent the sending of the message.

98269 UP When in command mode:

98270 Default.

98271 **APPLICATION USAGE**98272 Delivery of messages to remote systems requires the existence of communication paths to such
98273 systems. These need not exist.98274 Input lines are limited to {LINE_MAX} bytes, but mailers between systems may impose more
98275 severe line-length restrictions. This volume of POSIX.1-2017 does not place any restrictions on
98276 the length of messages handled by *mailx*, and for delivery of local messages the only limitations
98277 should be the normal problems of available disk space for the target mail file. When sending
98278 messages to external machines, applications are advised to limit messages to less than 100 000
98279 bytes because some mail gateways impose message-length restrictions.98280 The format of the system mailbox is intentionally unspecified. Not all systems implement
98281 system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some
98282 system mailboxes may be multiple files, others records in a database. The internal format of the
98283 messages themselves is specified with the historical format from Version 7, but only after the
98284 messages have been saved in some file other than the system mailbox. This was done so that
98285 many historical applications expecting text-file mailboxes are not broken.98286 Some new formats for messages can be expected in the future, probably including binary data,
98287 bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from
98288 handling such messages, but it must store them as text files in secondary mailboxes (unless some
98289 extension, such as a variable or command line option, is used to change the stored format). Its
98290 method of doing so is implementation-defined and might include translating the data into text
98291 file-compatible or readable form or omitting certain portions of the message from the stored
98292 output.98293 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**
98294 command discards all header-fields except those explicitly retained. The **discard** command
98295 keeps all header-fields except those explicitly discarded. If headers exist on the retained header

98296 list, **discard** and **ignore** commands are ignored.

98297 EXAMPLES

98298 None.

98299 RATIONALE

98300 The standard developers felt strongly that a method for applications to send messages to specific
98301 users was necessary. The obvious example is a batch utility, running non-interactively, that
98302 wishes to communicate errors or results to a user. However, the actual format, delivery
98303 mechanism, and method of reading the message are clearly beyond the scope of this volume of
98304 POSIX.1-2017.

98305 The intent of this command is to provide a simple, portable interface for sending messages non-
98306 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that
98307 implementations explicitly denote the sender and recipient in the body of the delivered message.
98308 Further specification of formats for either the message envelope or the message itself were
98309 deliberately not made, as the industry is in the midst of changing from the current standards to a
98310 more internationalized standard and it is probably incorrect, at this time, to require either one.

98311 Implementations are encouraged to conform to the various delivery mechanisms described in
98312 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request
98313 for Comment (RFC) documents RFC 819, RFC 920, RFC 921, RFC 1123, and RFC 5322 (which
98314 succeeded RFC 822).

98315 Many historical systems modified each body line that started with **From** by prefixing the 'F'
98316 with '>'. It is unnecessary, but allowed, to do that when the string does not follow a blank line
98317 because it cannot be confused with the next header.

98318 The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do
98319 not modify the contents of those messages in the mailbox; such a capability could be added as an
98320 extension, such as by using different command names.

98321 The restriction on a subject line being {LINE_MAX}-10 bytes is based on the historical format
98322 that consumes 10 bytes for **Subject:** and the trailing <newline>. Many historical mailers that a
98323 message may encounter on other systems are not able to handle lines that long, however.

98324 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient
98325 justification to exclude this utility from this volume of POSIX.1-2017. It is also arguable that it is,
98326 in fact, testable, but that the tests themselves are not portable.

98327 When *mailx* is being used by an application that wishes to receive the results as if none of the
98328 User Portability Utilities option features were supported, the *DEAD* environment variable must
98329 be set to */dev/null*. Otherwise, it may be subject to the file creations described in *mailx*
98330 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to
98331 */dev/null*, historical versions of *mailx* and *Mail* read initialization commands from a file before
98332 processing begins. Since the initialization that a user specifies could alter the contents of
98333 messages an application is trying to send, such applications must set *MAILRC* to */dev/null*.

98334 The description of *LC_TIME* uses “may affect” because many historical implementations do not
98335 or cannot manipulate the date and time strings in the incoming mail headers. Some headers
98336 found in incoming mail do not have enough information to determine the timezone in which the
98337 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal
98338 form that then is parsed by routines like *strftime()* that can take *LC_TIME* settings into account.
98339 Changing all these times to a user-specified format is allowed, but not required.

98340 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System
98341 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by

98342 declaring that *cat* is the paginator, would not meet with the intended meaning of this
 98343 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her
 98344 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike
 98345 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common
 98346 theme of user input, whereas editors differ dramatically.

98347 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to
 98348 be format details and were omitted.

98349 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them
 98350 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an
 98351 appropriate marketing distinction between systems.

98352 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*
 98353 option-argument to **-f**. By making *file* an operand, the guidelines are satisfied and users remain
 98354 portable. However, it does force implementations to support usage such as:

```
98355 mailx -fin mymail.box
```

98356 The **no name** method of unsetting variables is not present in all historical systems, but it is in
 98357 System V and provides a logical set of commands corresponding to the format of the display of
 98358 options from the *mailx set* command without arguments.

98359 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the
 98360 same feature. They are synonyms in this volume of POSIX.1-2017.

98361 The *mailx echo* command was not documented in the BSD version and has been omitted here
 98362 because it is not obviously useful for interactive users.

98363 The default prompt on the System V *mailx* is a <question-mark>, on BSD *Mail* an <ampersand>.
 98364 Since this volume of POSIX.1-2017 chose the *mailx* name, it kept the System V default, assuming
 98365 that BSD users would not have difficulty with this minor incompatibility (that they can
 98366 override).

98367 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,
 98368 since this volume of POSIX.1-2017 chose the *mailx* name, it kept the System V default, but allows
 98369 the SunOS user to achieve the desired results using **flpr**, an internal variable in System V *mailx*,
 98370 although it has not been documented in the SVID.

98371 The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command
 98372 escapes were adopted from 4.3 BSD *Mail*.

98373 The **version** command was not included because no sufficiently general specification of the
 98374 version information could be devised that would still be useful to a portable user. This
 98375 command name should be used by suppliers who wish to provide version information about the
 98376 *mailx* command.

98377 The “implementation-specific (unspecified) system start-up file” historically has been named
 98378 **/etc/mailx.rc**, but this specific name and location are not required.

98379 The intent of the wording for the **next** command is that if any command has already displayed
 98380 the current message it should display a following message, but, otherwise, it should display the
 98381 current message. Consider the command sequence:

```
98382 next 3  

  98383 delete 3  

  98384 next
```

98385 where the **autoprint** option was not set. The normative text specifies that the second **next**

98386 command should display a message following the third message, because even though the
98387 current message has not been displayed since it was set by the **delete** command, it has been
98388 displayed since the current message was anything other than message number 3. This does not
98389 always match historical practice in some implementations, where the command file address
98390 followed by **next** (or the default command) would skip the message for which the user had
98391 searched.

98392 **FUTURE DIRECTIONS**

98393 None.

98394 **SEE ALSO**

98395 [Chapter 2](#) (on page 2345), *ed*, *ls*, *more*, *vi*

98396 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

98397 **CHANGE HISTORY**

98398 First released in Issue 2.

98399 **Issue 5**

98400 The description of the *EDITOR* environment variable is changed to indicate that *ed* is the default
98401 editor if this variable is not set. In previous issues, this default was not stated explicitly at this
98402 point but was implied further down in the text.

98403 The FUTURE DIRECTIONS section is added.

98404 **Issue 6**

98405 The following new requirements on POSIX implementations derive from alignment with the
98406 Single UNIX Specification:

98407 The **-F** option is added.

98408 The **allnet**, **debug**, and **sendwait** internal variables are added.

98409 The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

98410 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “*HOME*
98411 directory” is replaced by “directory referred to by the *HOME* environment variable”.

98412 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which includes various
98413 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993
98414 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,
98415 #103, #106, #108, #114, #115, #122, and #129.

98416 The normative text is reworded to avoid use of the term “must” for application requirements.

98417 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

98418 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the
98419 EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was
98420 overlooked in the first version of this standard.

98421 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/17 is applied, updating the EXTENDED
98422 DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from
98423 “implementation-defined” to “unspecified” for consistency with other text in the EXTENDED
98424 DESCRIPTION.

98425 **Issue 7**

98426 Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC_TIME*
98427 environment variable.

98428 Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the **next**

98429 command.

98430 SD5-XCU-ERN-69 is applied.

98431 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

98432 Shading to indicate support for the User Portability Utilities option is added.

98433 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0120 [855] and XCU/TC2-2008/0121
98434 [619] are applied.

98435 **NAME**98436 `make` — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)98437 **SYNOPSIS**

```
98438 SD make [-einpqrst] [-f makefile]... [-k|-S] [macro=value...]
98439  [target_name...]
```

98440 **DESCRIPTION**

98441 The *make* utility shall update files that are derived from other files. A typical case is one where
 98442 object files are derived from the corresponding source files. The *make* utility examines time
 98443 relationships and shall update those derived files (called targets) that have modified times
 98444 earlier than the modified times of the files (called prerequisites) from which they are derived. A
 98445 description file (makefile) contains a description of the relationships between files, and the
 98446 commands that need to be executed to update the targets to reflect changes in their
 98447 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and
 98448 optional commands to be executed when a prerequisite is newer than the target. There are two
 98449 types of rule:

- 98450 1. *Inference rules*, which have one target name with at least one <period> ('.') and no
 98451 <slash> ('/')
- 98452 2. *Target rules*, which can have more than one target name

98453 In addition, *make* shall have a collection of built-in macros and inference rules that infer
 98454 prerequisite relationships to simplify maintenance of programs.

98455 To receive exactly the behavior described in this section, the user shall ensure that a portable
 98456 makefile shall:

98457 Include the special target **.POSIX**

98458 Omit any special target reserved for implementations (a leading period followed by
 98459 uppercase letters) that has not been specified by this section

98460 The behavior of *make* is unspecified if either or both of these conditions are not met.

98461 **OPTIONS**

98462 The *make* utility shall conform to XBD [Section 12.2](#) (on page 216), except for Guideline 9.

98463 The following options shall be supported:

98464 **-e** Cause environment variables, including those with null values, to override macro
 98465 assignments within makefiles.

98466 **-f *makefile*** Specify a different makefile. The argument *makefile* is a pathname of a description
 98467 file, which is also referred to as the *makefile*. A pathname of '-' shall denote the
 98468 standard input. There can be multiple instances of this option, and they shall be
 98469 processed in the order specified. The effect of specifying the same option-argument
 98470 more than once is unspecified.

98471 **-i** Ignore error codes returned by invoked commands. This mode is the same as if the
 98472 special target **.IGNORE** were specified without prerequisites.

98473 **-k** Continue to update other targets that do not depend on the current target if a non-
 98474 ignored error occurs while executing the commands to bring a target up-to-date.

98475 **-n** Write commands that would be executed on standard output, but do not execute
 98476 them. However, lines with a <plus-sign> ('+') prefix shall be executed. In this
 98477 mode, lines with an at-sign ('@') character prefix shall be written to standard

- 98478 output.
- 98479 **-p** Write to standard output the complete set of macro definitions and target
98480 descriptions. The output format is unspecified.
- 98481 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit
98482 value of 1. Targets shall not be updated if this option is specified. However, a
98483 makefile command line (associated with the targets) with a <plus-sign> ('+')
98484 prefix shall be executed.
- 98485 **-r** Clear the suffix list and do not use the built-in rules.
- 98486 **-S** Terminate *make* if an error occurs while executing the commands to bring a target
98487 up-to-date. This shall be the default and the opposite of **-k**.
- 98488 **-s** Do not write makefile command lines or touch messages (see **-t**) to standard
98489 output before executing. This mode shall be the same as if the special target
98490 **.SILENT** were specified without prerequisites.
- 98491 **-t** Update the modification time of each target as though a *touch target* had been
98492 executed. Targets that have prerequisites but no commands (see [Target Rules](#), on
98493 page 2974), or that are already up-to-date, shall not be touched in this manner.
98494 Write messages to standard output for each target file indicating the name of the
98495 file and that it was touched. Normally, the *makefile* command lines associated with
98496 each target are not executed. However, a command line with a <plus-sign> ('+')
98497 prefix shall be executed.

98498 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any
98499 options specified on the *make* utility command line. If the **-k** and **-S** options are both specified
98500 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option
98501 specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment
98502 variable, the result is undefined.

98503 OPERANDS

98504 The following operands shall be supported:

98505 *target_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is
98506 specified, while *make* is processing the makefiles, the first target that *make*
98507 encounters that is not a special target or an inference rule shall be used.

98508 *macro=value* Macro definitions, as defined in [Macros](#) (on page 2976).

98509 If the *target_name* and *macro=value* operands are intermixed on the *make* utility command line,
98510 the results are unspecified.

98511 STDIN

98512 The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT
98513 FILES section.

98514 INPUT FILES

98515 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,
98516 include lines, and comments. See the EXTENDED DESCRIPTION section.

98517 ENVIRONMENT VARIABLES

98518 The following environment variables shall affect the execution of *make*:

98519 *LANG* Provide a default value for the internationalization variables that are unset or null.
98520 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization
98521 variables used to determine the values of locale categories.)

98522		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
98523			
98524		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
98525			
98526			
98527		<i>LC_MESSAGES</i>	
98528			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
98529			
98530		<i>MAKEFLAGS</i>	
98531			This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):
98532			
98533			
98534			The characters are option letters without the leading <hyphen-minus> characters or <blank> separation used on a <i>make</i> utility command line.
98535			
98536			The characters are formatted in a manner similar to a portion of the <i>make</i> utility command line: options are preceded by <hyphen-minus> characters and <blank>-separated as described in XBD Section 12.2 (on page 216). The <i>macro=value</i> macro definition operands can also be included. The difference between the contents of <i>MAKEFLAGS</i> and the <i>make</i> utility command line is that the contents of the variable shall not be subjected to the word expansions (see Section 2.6 , on page 2353) associated with parsing the command line values.
98537			
98538			
98539			
98540			
98541			
98542			
98543			
98544	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
98545	XSI	<i>PROJECTDIR</i>	
98546			Provide a directory to be used to search for SCCS files not found in the current directory. In all of the following cases, the search for SCCS files is made in the directory SCCS in the identified directory. If the value of <i>PROJECTDIR</i> begins with a <slash>, it shall be considered an absolute pathname; otherwise, the value of <i>PROJECTDIR</i> is treated as a user name and that user's initial working directory shall be examined for a subdirectory src or source . If such a directory is found, it shall be used. Otherwise, the value is used as a relative pathname.
98547			
98548			
98549			
98550			
98551			
98552			
98553			If <i>PROJECTDIR</i> is not set or has a null value, the search for SCCS files shall be made in the directory SCCS in the current directory.
98554			
98555			The setting of <i>PROJECTDIR</i> affects all files listed in the remainder of this utility description for files with a component named SCCS .
98556			
98557			The value of the <i>SHELL</i> environment variable shall not be used as a macro and shall not be modified by defining the SHELL macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in Macros (on page 2976).
98558			
98559			
98560			
98561		ASYNCHRONOUS EVENTS	
98562			If not already ignored, <i>make</i> shall trap SIGHUP , SIGTERM , SIGINT , and SIGQUIT and remove the current target unless the target is a directory or the target is a prerequisite of the special target .PRECIOUS or unless one of the -n , -p , or -q options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, <i>make</i> shall take the standard action for all other signals.
98563			
98564			
98565			
98566			
98567			

98568 **STDOUT**

98569 The *make* utility shall write all commands to be executed to standard output unless the `-s` option
 98570 was specified, the command is prefixed with an at-sign, or the special target **.SILENT** has either
 98571 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work
 98572 needing to be done, it shall write a message to standard output indicating that no action was
 98573 taken. If the `-t` option is present and a file is touched, *make* shall write to standard output a
 98574 message of unspecified format indicating that the file was touched, including the filename of the
 98575 file.

98576 **STDERR**

98577 The standard error shall be used only for diagnostic messages.

98578 **OUTPUT FILES**

98579 Files can be created when the `-t` option is present. Additional files can also be created by the
 98580 utilities invoked by *make*.

98581 **EXTENDED DESCRIPTION**

98582 The *make* utility attempts to perform the actions required to ensure that the specified targets are
 98583 up-to-date. A target shall be considered up-to-date if it exists and is newer than all of its
 98584 dependencies, or if it has already been made up-to-date by the current invocation of *make*
 98585 (regardless of the target's existence or age). A target may also be considered up-to-date if it
 98586 exists, is the same age as one or more of its prerequisites, and is newer than the remaining
 98587 prerequisites (if any). The *make* utility shall treat all prerequisites as targets themselves and
 98588 recursively ensure that they are up-to-date, processing them in the order in which they appear in
 98589 the rule. The *make* utility shall use the modification times of files to determine whether the
 98590 corresponding targets are out-of-date.

98591 To ensure that a target is up-to-date, *make* shall ensure that all of the prerequisites of a target are
 98592 up-to-date, then check to see if the target itself is up-to-date. If the target is not up-to-date, the
 98593 target shall be made up-to-date by executing the rule's commands (if any). If the target does not
 98594 exist after the target has been successfully made up-to-date, the target shall be treated as being
 98595 newer than any target for which it is a prerequisite.

98596 If a target exists and there is neither a target rule nor an inference rule for the target, the target
 98597 shall be considered up-to-date. It shall be an error if *make* attempts to ensure that a target is up-
 98598 to-date but the target does not exist and there is neither a target rule nor an inference rule for the
 98599 target.

98600 **Makefile Syntax**

98601 A makefile can contain rules, macro definitions (see [Macros](#), on page 2976), include lines, and
 98602 comments. There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall
 98603 contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be
 98604 used and the suffix list shall be cleared. Additional rules of both types can be specified in a
 98605 makefile. If a rule is defined more than once, the value of the rule shall be that of the last one
 98606 specified. Macros can also be defined more than once, and the value of the macro is specified in
 98607 [Macros](#) (on page 2976). There are three kinds of comments: blank lines, empty lines, and a
 98608 <number-sign> ('#') and all following characters up to the first unescaped <newline>
 98609 character. Blank lines, empty lines, and lines with <number-sign> ('#') as the first character on
 98610 the line are also known as comment lines.

98611 By default, the following files shall be tried in sequence: **./makefile** and **./Makefile**. If neither
 98612 XSI **./makefile** or **./Makefile** are found, other implementation-defined files may also be tried. On
 98613 XSI-conformant systems, the additional files **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and
 98614 **SCCS/s.Makefile** shall also be tried.

98615 The `-f` option shall direct *make* to ignore any of these default files and use the specified argument
98616 as a makefile instead. If the `-` argument is specified, standard input shall be used.

98617 The term *makefile* is used to refer to any rules provided by the user, whether in `./makefile` or its
98618 variants, or specified by the `-f` option.

98619 The rules in makefiles shall consist of the following types of lines: target rules, including special
98620 targets (see [Target Rules](#), on page 2974), inference rules (see [Inference Rules](#), on page 2977),
98621 macro definitions (see [Macros](#), on page 2976), and comments.

98622 Target and Inference Rules may contain *command lines*. Command lines can have a prefix that
98623 shall be removed before execution (see [Makefile Execution](#), on page 2974).

98624 When an escaped `<newline>` (one preceded by a `<backslash>`) is found anywhere in the
98625 makefile except in a command line, an include line, or a line immediately preceding an include
98626 line, it shall be replaced, along with any leading white space on the following line, with a single
98627 `<space>`. When an escaped `<newline>` is found in a command line in a makefile, the command
98628 line shall contain the `<backslash>`, the `<newline>`, and the next line, except that the first
98629 character of the next line shall not be included if it is a `<tab>`. When an escaped `<newline>` is
98630 found in an include line or in a line immediately preceding an include line, the behavior is
98631 unspecified.

98632 **Include Lines**

98633 If the word **include** appears at the beginning of a line and is followed by one or more `<blank>`
98634 characters, the string formed by the remainder of the line shall be processed as follows to
98635 produce a pathname:

98636 The trailing `<newline>`, any `<blank>` characters immediately preceding a comment, and
98637 any comment shall be discarded. If the resulting string contains any double-quote
98638 characters (`'"'`) the behavior is unspecified.

98639 The resulting string shall be processed for macro expansion (see [Macros](#), on page 2976).

98640 Any `<blank>` characters that appear after the first non-`<blank>` shall be used as separators
98641 to divide the macro-expanded string into fields. It is unspecified whether any other white-
98642 space characters are also used as separators. It is unspecified whether pathname expansion
98643 (see [Section 2.13](#), on page 2382) is also performed.

98644 If the processing of separators and optional pathname expansion results in either zero or
98645 two or more non-empty fields, the behavior is unspecified. If it results in one non-empty
98646 field, that field is taken as the pathname.

98647 If the pathname does not begin with a `/'` it shall be treated as relative to the current working
98648 directory of the process, not relative to the directory containing the makefile. If the file does not
98649 exist in this location, it is unspecified whether additional directories are searched.

98650 The contents of the file specified by the pathname shall be read and processed as if they
98651 appeared in the makefile in place of the include line. If the file ends with an escaped `<newline>`
98652 the behavior is unspecified.

98653 The file may itself contain further include lines. Implementations shall support nesting of
98654 include files up to a depth of at least 16.

98655 **Makefile Execution**

98656 Makefile command lines shall be processed one at a time.

98657 Makefile command lines can have one or more of the following prefixes: a <hyphen-minus>
98658 ('-'), an at-sign ('@'), or a <plus-sign> ('+'). These shall modify the way in which *make*
98659 processes the command.98660 – If the command prefix contains a <hyphen-minus>, or the `-i` option is present, or the special
98661 target `.IGNORE` has either the current target as a prerequisite or has no prerequisites, any
98662 error found while executing the command shall be ignored.98663 @ If the command prefix contains an at-sign and the *make* utility command line `-n` option is
98664 not specified, or the `-s` option is present, or the special target `.SILENT` has either the current
98665 target as a prerequisite or has no prerequisites, the command shall not be written to
98666 standard output before it is executed.98667 + If the command prefix contains a <plus-sign>, this indicates a makefile command line that
98668 shall be executed even if `-n`, `-q`, or `-t` is specified.98669 An *execution line* is built from the command line by removing any prefix characters. Except as
98670 described under the at-sign prefix, the execution line shall be written to the standard output,
98671 optionally preceded by a <tab>. The execution line shall then be executed by a shell as if it were
98672 passed as the argument to the *system()* interface, except that if errors are not being ignored then
98673 the shell `-e` option shall also be in effect. If errors are being ignored for the command (as a result
98674 of the `-i` option, a '-' command prefix, or a `.IGNORE` special target), the shell `-e` option shall
98675 not be in effect. The environment for the command being executed shall contain all of the
98676 variables in the environment of *make*.98677 By default, when *make* receives a non-zero status from the execution of a command, it shall
98678 terminate with an error message to standard error.98679 **Target Rules**

98680 Target rules are formatted as follows:

98681 `target [target...]: [prerequisite...][;command]`
98682 `[<tab>command`
98683 `<tab>command`
98684 `...]`98685 *line that does not begin with <tab>*98686 Target entries are specified by a <blank>-separated, non-null list of targets, then a <colon>, then
98687 a <blank>-separated, possibly empty list of prerequisites. Text following a <semicolon>, if any,
98688 and all following lines that begin with a <tab>, are makefile command lines to be executed to
98689 update the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a
98690 new entry. Any comment line may begin a new entry.98691 Applications shall select target names from the set of characters consisting solely of periods,
98692 underscores, digits, and alphabets from the portable character set (see XBD [Section 6.1](#), on
98693 page 125). Implementations may allow other characters in target names as extensions. The
98694 interpretation of targets containing the characters '%' and '"' is implementation-defined.98695 A target that has prerequisites, but does not have any commands, can be used to add to the
98696 prerequisite list for that target. Only one target rule for any given target can contain commands.98697 Lines that begin with one of the following are called *special targets* and control the operation of
98698 *make*:

- 98699 **.DEFAULT** If the makefile uses this special target, the application shall ensure that it is
 98700 specified with commands, but without prerequisites. The commands shall be used
 98701 by *make* if there are no other rules available to build a target.
- 98702 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors
 98703 from commands associated with them to be ignored in the same manner as
 98704 specified by the `-i` option. Subsequent occurrences of **.IGNORE** shall add to the
 98705 list of targets ignoring command errors. If no prerequisites are specified, *make* shall
 98706 behave as if the `-i` option had been specified and errors from all commands
 98707 associated with all targets shall be ignored.
- 98708 **.POSIX** The application shall ensure that this special target is specified without
 98709 prerequisites or commands. If it appears as the first non-comment line in the
 98710 makefile, *make* shall process the makefile as specified by this section; otherwise, the
 98711 behavior of *make* is unspecified.
- 98712 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the
 98713 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS
 98714 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious
 98715 files. If no prerequisites are specified, all targets in the makefile shall be treated as
 98716 if specified with **.PRECIOUS**.
- 98717 XSI **.SCCS_GET** The application shall ensure that this special target is specified without
 98718 prerequisites. If this special target is included in a makefile, the commands
 98719 specified with this target shall replace the default commands associated with this
 98720 special target (see [Default Rules](#), on page 2980). The commands specified with this
 98721 target are used to get all SCCS files that are not found in the current directory.
- 98722 When source files are named in a dependency list, *make* shall treat them just like
 98723 any other target. Because the source file is presumed to be present in the directory,
 98724 there is no need to add an entry for it to the makefile. When a target has no
 98725 dependencies, but is present in the directory, *make* shall assume that that file is up-
 98726 to-date. If, however, an SCCS file named **SCCS/s.source_file** is found for a target
 98727 *source_file*, *make* compares the timestamp of the target file with that of the
 98728 **SCCS/s.source_file** to ensure the target is up-to-date. If the target is missing, or if
 98729 the SCCS file is newer, *make* shall automatically issue the commands specified for
 98730 the **.SCCS_GET** special target to retrieve the most recent version. However, if the
 98731 target is writable by anyone, *make* shall not retrieve a new version.
- 98732 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause
 98733 commands associated with them not to be written to the standard output before
 98734 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of
 98735 targets with silent commands. If no prerequisites are specified, *make* shall behave
 98736 as if the `-s` option had been specified and no commands or touch messages
 98737 associated with any target shall be written to standard output.
- 98738 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are
 98739 used in conjunction with the inference rules (see [Inference Rules](#), on page 2977). If
 98740 **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be
 98741 cleared.
- 98742 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified
 98743 without commands.
- 98744 Targets with names consisting of a leading <period> followed by the uppercase letters "POSIX"
 98745 and then any other characters are reserved for future standardization. Targets with names

98746 consisting of a leading <period> followed by one or more uppercase letters are reserved for
98747 implementation extensions.

98748 **Macros**

98749 Macro definitions are in the form:

```
98750 string1 = [string2]
```

98751 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all
98752 characters, if any, after the <equals-sign>, up to a comment character ('#') or an unescaped
98753 <newline>. Any <blank> characters immediately before or after the <equals-sign> shall be
98754 ignored.

98755 Applications shall select macro names from the set of characters consisting solely of periods,
98756 underscores, digits, and alphabets from the portable character set (see XBD [Section 6.1](#), on
98757 page 125). A macro name shall not contain an <equals-sign>. Implementations may allow other
98758 characters in macro names as extensions.

98759 Macros can appear anywhere in the makefile. Macro expansions using the forms $\$(string1)$ or
98760 $\${string1}$ shall be replaced by *string2*, as follows:

98761 Macros in target lines shall be evaluated when the target line is read.

98762 Macros in makefile command lines shall be evaluated when the command is executed.

98763 Macros in the string before the <equals-sign> in a macro definition shall be evaluated
98764 when the macro assignment is made.

98765 Macros after the <equals-sign> in a macro definition shall not be evaluated until the
98766 defined macro is used in a rule or command, or before the <equals-sign> in a macro
98767 definition.

98768 The parentheses or braces are optional if *string1* is a single character. The macro \$\$ shall be
98769 replaced by the single character '\$'. If *string1* in a macro expansion contains a macro
98770 expansion, the results are unspecified.

98771 Macro expansions using the forms $\$(string1[:subst1=[subst2]])$ or $\${string1[:subst1=[subst2]]}$ can
98772 be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is
98773 performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word
98774 in *string1* (where a *word*, in this context, is defined to be a string delimited by the beginning of
98775 the line, a <blank>, or a <newline>). If *string1* in a macro expansion contains a macro expansion,
98776 the results are unspecified. If a <percent-sign> character appears as part of *subst1* or *subst2* after
98777 any macros have been recursively expanded, the results are unspecified.

98778 Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro
98779 expansions in *string2* of macro definition lines shall be performed when the macro identified by
98780 *string1* is expanded in a rule or command.

98781 Macro definitions shall be taken from the following sources, in the following logical order,
98782 before the makefile(s) are read.

- 98783 1. Macros specified on the *make* utility command line, in the order specified on the
98784 command line. It is unspecified whether the internal macros defined in [Internal Macros](#)
98785 (on page 2979) are accepted from this source.
- 98786 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the
98787 environment variable. It is unspecified whether the internal macros defined in [Internal](#)
98788 [Macros](#) (on page 2979) are accepted from this source.

98789 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and
98790 including the variables with null values.

98791 4. Macros defined in the inference rules built into *make*.

98792 Macro definitions from these sources shall not override macro definitions from a lower-
98793 numbered source. Macro definitions from a single source (for example, the *make* utility
98794 command line, the *MAKEFLAGS* environment variable, or the other environment variables)
98795 shall override previous macro definitions from the same source.

98796 Macros defined in the makefile(s) shall override macro definitions that occur before them in the
98797 makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined
98798 in the makefile(s) shall override macro definitions from source 3. Macros defined in the
98799 makefile(s) shall not override macro definitions from source 1 or source 2.

98800 Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*)
98801 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not
98802 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted
98803 in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance
98804 of the *make* command, the original macro's value is recovered. Other implementation-defined
98805 options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of
98806 the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the
98807 *MAKEFLAGS* environment variable shall be modified to match the new value of the
98808 *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

98809 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the
98810 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other
98811 implementation-defined variables may also be added to the environment of *make*. Macros
98812 defined by the *MAKEFLAGS* environment variable and macros defined in the makefile(s) shall
98813 not be added to the environment of *make* if they are not already in its environment. With the
98814 exception of *SHELL* (see below), it is unspecified whether macros defined in these ways update
98815 the value of an environment variable that already exists in the environment of *make*.

98816 The **SHELL** macro shall be treated specially. It shall be provided by *make* and set to the
98817 pathname of the shell command language interpreter (see *sh*). The *SHELL* environment variable
98818 shall not affect the value of the **SHELL** macro. If **SHELL** is defined in the makefile or is specified
98819 on the command line, it shall replace the original value of the **SHELL** macro, but shall not affect
98820 the *SHELL* environment variable. Other effects of defining **SHELL** in the makefile or on the
98821 command line are implementation-defined.

98822 Inference Rules

98823 Inference rules are formatted as follows:

```
98824 target:
98825 <tab>command
98826 [<tab>command]
98827 ...
98828 line that does not begin with <tab> or #
```

98829 The application shall ensure that the *target* portion is a valid target name (see [Target Rules](#), on
98830 page 2974) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as
98831 prerequisites of the **SUFFIXES** special target and *s1* and *s2* do not contain any *<slash>* or
98832 *<period>* characters.) If there is only one *<period>* in the target, it is a single-suffix inference
98833 rule. Targets with two periods are double-suffix inference rules. Inference rules can have only
98834 one target before the *<colon>*.

98835 The application shall ensure that the makefile does not specify prerequisites for inference rules;
 98836 no characters other than white space shall follow the <colon> in the first line, except when
 98837 creating the *empty rule*, described below. Prerequisites are inferred, as described below.

98838 Inference rules can be redefined. A target that matches an existing inference rule shall overwrite
 98839 the old inference rule. An empty rule can be created with a command consisting of simply a
 98840 <semicolon> (that is, the rule still exists and is found during inference rule search, but since it is
 98841 empty, execution has no effect). The empty rule can also be formatted as follows:

```
98842 rule: ;
```

98843 where zero or more <blank> characters separate the <colon> and <semicolon>.

98844 The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be
 98845 made up-to-date. A list of inference rules defines the commands to be executed. By default, *make*
 98846 contains a built-in set of inference rules. Additional rules can be specified in the makefile.

98847 The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by
 98848 the inference rules. The order in which the suffixes are specified defines the order in which the
 98849 inference rules for the suffixes are used. New suffixes shall be appended to the current list by
 98850 specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites
 98851 shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is
 98852 required to change the order of the suffixes.

98853 Normally, the user would provide an inference rule for each suffix. The inference rule to update
 98854 a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The
 98855 internal macros provide the means to specify general inference rules (see [Internal Macros](#), on
 98856 page 2979).

98857 When no target rule is found to update a target, the inference rules shall be checked. The suffix
 98858 of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special
 98859 targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order
 98860 defined for the first **.s2.s1** rule whose prerequisite file (**\$.s2**) exists. If the target is out-of-date
 98861 with respect to this prerequisite, the commands for that inference rule shall be executed.

98862 If the target to be built does not contain a suffix and there is no rule for the target, the single
 98863 suffix inference rules shall be checked. The single-suffix inference rules define how to build a
 98864 target if a file is found with a name that matches the target name with one of the single suffixes
 98865 appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The
 98866 other suffix (**.s1**) is treated as null.

98867 XSI A <tilde> ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule
 98868 **.c~.o** would transform an SCCS C-language source file into an object file (**.o**). Because the **s.**
 98869 of the SCCS files is a prefix, it is incompatible with *make's* suffix point of view. Hence, the '~' is a
 98870 way of changing any file reference into an SCCS file reference.

98871 Libraries

98872 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive
 98873 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*
 98874 to the member name. The application shall ensure that the member is an object file with the **.o**
 98875 suffix. The modification time of the expression is the modification time for the member as kept
 98876 in the archive library; see *ar*. The **.a** suffix shall refer to an archive library. The **.s2.a** rule shall be
 98877 used to update a member in the library from a file with a suffix **.s2**.

98878 **Internal Macros**

98879 The *make* utility shall maintain five internal macros that can be used in target and inference rules.
 98880 In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,
 98881 *inference rule*, *target*, and *prerequisite* is necessary.

98882 Target rules are specified by the user in a makefile for a particular target. Inference rules are
 98883 user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites
 98884 are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those
 98885 prerequisites that are generated when inference rules are used. Inference rules are applied to
 98886 implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in
 98887 the makefile. Target rules are applied to targets specified in the makefile.

98888 Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)
 98889 shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon
 98890 recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be
 98891 processed recursively until a target is found that has no prerequisites, or further recursion would
 98892 require applying two inference rules one immediately after the other, at which point the
 98893 recursion shall stop. As an extension, implementations may continue recursion when two or
 98894 more successive inference rules need to be applied; however, if there are multiple different
 98895 chains of such rules that could be used to create the target, it is unspecified which chain is used.
 98896 The recursion shall then back up, updating each target as it goes.

98897 In the definitions that follow, the word *target* refers to one of:

98898 A target specified in the makefile

98899 An explicit prerequisite specified in the makefile that becomes the target when *make*
 98900 processes it during recursion

98901 An implicit prerequisite that becomes a target when *make* processes it during recursion

98902 In the definitions that follow, the word *prerequisite* refers to one of the following:

98903 An explicit prerequisite specified in the makefile for a particular target

98904 An implicit prerequisite generated as a result of locating an appropriate inference rule and
 98905 corresponding file that matches the suffix of the target

98906 The five internal macros are:

98907 **\$@** The **\$@** shall evaluate to the full target name of the current target, or the archive
 98908 filename part of a library archive target. It shall be evaluated for both target and
 98909 inference rules.

98910 For example, in the **.c.a** inference rule, **\$@** represents the out-of-date **.a** file to be built.
 98911 Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-
 98912 date **lib.a**.

98913 **\$\$** The **\$\$** macro shall be evaluated only when the current target is an archive library
 98914 member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and
 98915 **\$\$** shall evaluate to *member.o*. The **\$\$** macro shall be evaluated for both target and
 98916 inference rules.

98917 For example, in a makefile target rule to build **lib.a(file.o)**, **\$\$** represents **file.o**, as
 98918 opposed to **\$@**, which represents **lib.a**.

98919 **\$\$?** The **\$\$?** macro shall evaluate to the list of prerequisites that are newer than the current
 98920 target. It shall be evaluated for both target and inference rules.

98921 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
 98922 where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to
 98923 **file2.o** and **file3.o**, `$$?` represents **file2.o** and **file3.o**.

98924 `$$<` In an inference rule, the `$$<` macro shall evaluate to the filename whose existence
 98925 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the `$$<`
 98926 macro shall evaluate to the current target name. The meaning of the `$$<` macro shall be
 98927 otherwise unspecified.

98928 For example, in the **.c.a** inference rule, `$$<` represents the prerequisite **.c** file.

98929 `$$*` The `$$*` macro shall evaluate to the current target name with its suffix deleted. It shall be
 98930 evaluated at least for inference rules.

98931 For example, in the **.c.a** inference rule, `$$*.o` represents the out-of-date **.o** file that
 98932 corresponds to the prerequisite **.c** file.

98933 Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended
 98934 to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part*
 98935 for 'F'. The directory part is the path prefix of the file without a trailing `<slash>`; for the current
 98936 directory, the directory part is `'.'`. When the `$$?` macro contains more than one prerequisite
 98937 filename, the `$$(?D)` and `$$(?F)` (or `$${?D}` and `$${?F}`) macros expand to a list of directory name parts
 98938 and filename parts respectively.

98939 For the target *lib(member.o)* and the **s2.a** rule, the internal macros shall be defined as:

98940 `$$<` *member.s2*

98941 `$$*` *member*

98942 `$$@` *lib*

98943 `$$?` *member.s2*

98944 `$$%` *member.o*

98945 Default Rules

98946 The default rules for *make* shall achieve results that are the same as if the following were used.
 98947 Implementations that do not support the C-Language Development Utilities option may omit
 98948 **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.
 98949 Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference
 98950 rules. Implementations may provide additional macros and rules.

98951 SPECIAL TARGETS

98952 XSI `.SCCS_GET: sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $$@`

98953 XSI `.SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~`

98954 MACROS

98955 `MAKE=make`

98956 `AR=ar`

98957 `ARFLAGS=-rv`

98958 `YACC=yacc`

98959 `YFLAGS=`

98960 `LEX=lex`

98961 `LFLAGS=`

```

98962     LDFLAGS=
98963     CC=c99
98964     CFLAGS=-O 1
98965     FC=fort77
98966     FFLAGS=-O 1
98967 XSI   GET=get
98968     GFLAGS=
98969     SCCSFLAGS=
98970     SCCSGETFLAGS=-s

```

98971 *SINGLE SUFFIX RULES*

```

98972     .c:
98973         $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<
98974     .f:
98975         $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<
98976     .sh:
98977         cp $< $@
98978         chmod a+x $@

```

```

98979 XSI   .c~:
98980         $(GET) $(GFLAGS) -p $< > $*.c
98981         $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
98982     .f~:
98983         $(GET) $(GFLAGS) -p $< > $*.f
98984         $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
98985     .sh~:
98986         $(GET) $(GFLAGS) -p $< > $*.sh
98987         cp $*.sh $@
98988         chmod a+x $@

```

98989 *DOUBLE SUFFIX RULES*

```

98990     .c.o:
98991         $(CC) $(CFLAGS) -c $<
98992     .f.o:
98993         $(FC) $(FFLAGS) -c $<
98994     .y.o:
98995         $(YACC) $(YFLAGS) $<
98996         $(CC) $(CFLAGS) -c y.tab.c
98997         rm -f y.tab.c
98998         mv y.tab.o $@
98999     .l.o:
99000         $(LEX) $(LFLAGS) $<
99001         $(CC) $(CFLAGS) -c lex.yy.c
99002         rm -f lex.yy.c
99003         mv lex.yy.o $@
99004     .y.c:

```

```

99005         $(YACC) $(YFLAGS) $<
99006         mv y.tab.c $@

99007     .l.c:
99008         $(LEX) $(LFLAGS) $<
99009         mv lex.yy.c $@

99010 XSI     .c~.o:
99011         $(GET) $(GFLAGS) -p $< > $*.c
99012         $(CC) $(CFLAGS) -c $*.c

99013     .f~.o:
99014         $(GET) $(GFLAGS) -p $< > $*.f
99015         $(FC) $(FFLAGS) -c $*.f

99016     .y~.o:
99017         $(GET) $(GFLAGS) -p $< > $*.y
99018         $(YACC) $(YFLAGS) $*.y
99019         $(CC) $(CFLAGS) -c y.tab.c
99020         rm -f y.tab.c
99021         mv y.tab.o $@

99022     .l~.o:
99023         $(GET) $(GFLAGS) -p $< > $*.l
99024         $(LEX) $(LFLAGS) $*.l
99025         $(CC) $(CFLAGS) -c lex.yy.c
99026         rm -f lex.yy.c
99027         mv lex.yy.o $@

99028     .y~.c:
99029         $(GET) $(GFLAGS) -p $< > $*.y
99030         $(YACC) $(YFLAGS) $*.y
99031         mv y.tab.c $@

99032     .l~.c:
99033         $(GET) $(GFLAGS) -p $< > $*.l
99034         $(LEX) $(LFLAGS) $*.l
99035         mv lex.yy.c $@

```

```

99036     .c.a:
99037         $(CC) -c $(CFLAGS) $<
99038         $(AR) $(ARFLAGS) $@ $*.o
99039         rm -f $*.o

```

```

99040     .f.a:
99041         $(FC) -c $(FFLAGS) $<
99042         $(AR) $(ARFLAGS) $@ $*.o
99043         rm -f $*.o

```

99044 EXIT STATUS

99045 When the `-q` option is specified, the *make* utility shall exit with one of the following values:

- 99046 0 Successful completion.
- 99047 1 The target was not up-to-date.

99048 >1 An error occurred.

99049 When the `-q` option is not specified, the *make* utility shall exit with one of the following values:

99050 0 Successful completion.

99051 >0 An error occurred.

99052 CONSEQUENCES OF ERRORS

99053 Default.

99054 APPLICATION USAGE

99055 If there is a source file (such as *./source.c*) and there are two SCCS files corresponding to it
 99056 (*./s.source.c* and *./SCCS/s.source.c*), on XSI-conformant systems *make* uses the SCCS file in the
 99057 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,
 99058 *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for
 99059 a given source file, future developers are very likely to be confused.

99060 It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to
 99061 guarantee that they are not affected by local extensions.

99062 The `-k` and `-S` options are both present so that the relationship between the command line, the
 99063 *MAKEFLAGS* variable, and the makefile can be controlled precisely. If the `k` flag is passed in
 99064 *MAKEFLAGS* and a command is of the form:

```
99065 $(MAKE) -S foo
```

99066 then the default behavior is restored for the child *make*.

99067 When the `-n` option is specified, it is always added to *MAKEFLAGS*. This allows a recursive
 99068 *make -n target* to be used to see all of the action that would be taken to update *target*.

99069 Because of widespread historical practice, interpreting a `<number-sign>` ('#') inside a variable
 99070 as the start of a comment has the unfortunate side-effect of making it impossible to place a
 99071 `<number-sign>` in a variable, thus forbidding something like:

```
99072 CFLAGS = "-D COMMENT_CHAR='#'"
```

99073 Many historical *make* utilities stop chaining together inference rules when an intermediate target
 99074 is nonexistent. For example, it might be possible for a *make* to determine that both *.y.c* and *.c.o*
 99075 could be used to convert a *.y* to a *.o*. Instead, in this case, *make* requires the use of a *.y.o* rule.

99076 The best way to provide portable makefiles is to include all of the rules needed in the makefile
 99077 itself. The rules provided use only features provided by other parts of this volume of
 99078 POSIX.1-2017. The default rules include rules for optional commands in this volume of
 99079 POSIX.1-2017. Only rules pertaining to commands that are provided are needed in an
 99080 implementation's default set.

99081 Macros used within other macros are evaluated when the new macro is used rather than when
 99082 the new macro is defined. Therefore:

```
99083 MACRO = value1
99084 NEW   = $(MACRO)
99085 MACRO = value2
```

```
99086 target:
99087     echo $(NEW)
```

99088 would produce *value2* and not *value1* since **NEW** was not expanded until it was needed in the
 99089 *echo* command line.

99090 Some historical applications have been known to intermix *target_name* and *macro=name* operands
 99091 on the command line, expecting that all of the macros are processed before any of the targets are
 99092 dealt with. Conforming applications do not do this, although some backwards-compatibility
 99093 support may be included in some implementations.

99094 The following characters in filenames may give trouble: '=', ':', '`', single-quote, and '@'.
 99095 In include filenames, pattern matching characters and '"' should also be avoided, as they may
 99096 be treated as special by some implementations.

99097 For inference rules, the description of \$< and \$? seem similar. However, an example shows the
 99098 minor difference. In a makefile containing:

```
99099 foo.o: foo.h
```

99100 if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from
 99101 **foo.c** is used, with \$< equal to **foo.c** and \$? equal to **foo.h**. If **foo.c** is also newer than **foo.o**, \$<
 99102 is equal to **foo.c** and \$? is equal to **foo.h** **foo.c**.

99103 As a consequence of the general rules for target updating, a useful special case is that if a target
 99104 has no prerequisites and no commands, and the target of the rule is a nonexistent file, then *make*
 99105 acts as if this target has been updated whenever its rule is run.

99106 **Note:** This implies that all targets depending on this one will always have their commands run.

99107 Shell command sequences like `make; cp original copy; make` may have problems on
 99108 filesystems where the timestamp resolution is the minimum (1 second) required by the standard
 99109 and where *make* considers identical timestamps to be up-to-date. Conversely, rules like
 99110 `copy: original; cp -p original copy` will result in redundant work on *make*
 99111 implementations that consider identical timestamps to be out-of-date.

99112 This standard does not specify precedence between macro definition and include directives.
 99113 Thus, the behavior of:

```
99114 include =foo.mk
```

99115 is unspecified. To define a variable named `include`, either the white space before the <equal-
 99116 sign> should be removed, or another macro should be used, as in:

```
99117 INCLUDE_NAME = include  

  99118 $(INCLUDE_NAME) =foo.mk
```

99119 On the other hand, if the intent is to include a file which starts with an <equal-sign>, either the
 99120 filename should be changed to `./=foo.mk`, or the makefile should be written as:

```
99121 INCLUDE_FILE = =foo.mk  

  99122 include $(INCLUDE_FILE)
```

99123 EXAMPLES

99124 1. The following command:

```
99125 make
```

99126 makes the first target found in the makefile.

99127 2. The following command:

```
99128 make junk
```

99129 makes the target **junk**.

99130 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in
 99131 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

```
99132 .POSIX:
99133 pgm: a.o b.o
99134     c99 a.o b.o -o pgm
99135 a.o: incl.h a.c
99136     c99 -c a.c
99137 b.o: incl.h b.c
99138     c99 -c b.c
```

99139 4. An example for making optimized **.o** files from **.c** files is:

```
99140 .c.o:
99141     c99 -c -O 1 $*.c

99142 or:

99143 .c.o:
99144     c99 -c -O 1 $<
```

99145 5. The most common use of the archive interface follows. Here, it is assumed that the source
 99146 files are all C-language source:

```
99147 lib: lib(file1.o) lib(file2.o) lib(file3.o)
99148     @echo lib is now up-to-date
```

99149 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.

99150 The treatment of escaped <newline> characters throughout the makefile is historical
 99151 practice. For example, the inference rule:

```
99152 .c.o\  

99153 :
```

99154 works, and the macro:

```
99155 f= bar baz\  

99156     biz
99157 a:
99158     echo ==$f==
```

99159 echoes "==bar baz biz==".

99160 If **\$?** were:

```
99161 /usr/include/stdio.h /usr/include/unistd.h foo.h
```

99162 then **\$(?D)** would be:

```
99163 /usr/include /usr/include .
```

99164 and **\$(?F)** would be:

```
99165 stdio.h unistd.h foo.h
```

99166 6. The contents of the built-in rules can be viewed by running:

```
99167 make -p -f /dev/null 2>/dev/null
```


99168 **RATIONALE**

99169 The *make* utility described in this volume of POSIX.1-2017 is intended to provide the means for
 99170 changing portable source code into executables that can be run on an POSIX.1-2017-conforming
 99171 system. It reflects the most common features present in System V and BSD *makes*.

99172 Historically, the *make* utility has been an especially fertile ground for vendor and research
 99173 organization-specific syntax modifications and extensions. Examples include:

99174 Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,
 99175 and others)

99176 Additional “operators” separating targets and their prerequisites (System V, BSD, and
 99177 others)

99178 Specifying that command lines containing the strings “`{MAKE}`” and “`$(MAKE)`” are
 99179 executed when the `-n` option is specified (GNU and System V)

99180 Modifications of the meaning of internal macros when referencing libraries (BSD and
 99181 others)

99182 Using a single instance of the shell for all of the command lines of the target (BSD and
 99183 others)

99184 Allowing `<space>` characters as well as `<tab>` characters to delimit command lines (BSD)

99185 Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and
 99186 others)

99187 Remote execution of command lines (Sprite and others)

99188 Specifying additional special targets (BSD, System V, and most others)

99189 Specifying an alternate shell to use to process commands.

99190 Additionally, many vendors and research organizations have rethought the basic concepts of
 99191 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of
 99192 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of
 99193 POSIX.1-2017 to require behavior that would be incompatible (and probably inferior) to
 99194 historical practice for such a community.

99195 In similar circumstances, when the industry has enough sufficiently incompatible formats as to
 99196 make them irreconcilable, this volume of POSIX.1-2017 has followed one or both of two courses
 99197 of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line options
 99198 have been provided to select the desired behavior (*grep*, *od*, and *pax*).

99199 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes
 99200 accepted by almost all versions of *make*, it was decided that it would be counter-productive to
 99201 change the name. And since the makefile itself is a basic unit of portability, it would not be
 99202 completely effective to reserve a new option letter, such as *make -P*, to achieve the portable
 99203 behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to
 99204 specify “standard” behavior. This special target does not preclude extensions in the *make* utility,
 99205 nor does it preclude such extensions being used by the makefile specifying the target; it does,
 99206 however, preclude any extensions from being applied that could alter the behavior of previously
 99207 valid syntax; such extensions must be controlled via command line options or new special
 99208 targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to
 99209 guarantee that they are not affected by local extensions.

99210 The portable version of *make* described in this reference page is not intended to be the state-of-
 99211 the-art software generation tool and, as such, some newer and more leading-edge features have

99212 not been included. An attempt has been made to describe the portable makefile in a manner that
 99213 does not preclude such extensions as long as they do not disturb the portable behavior described
 99214 here.

99215 When the `-n` option is specified, it is always added to `MAKEFLAGS`. This allows a recursive
 99216 `make -n target` to be used to see all of the action that would be taken to update `target`.

99217 The definition of `MAKEFLAGS` allows both the System V letter string and the BSD command
 99218 line formats. The two formats are sufficiently different to allow implementations to support both
 99219 without ambiguity.

99220 Early proposals stated that an “unquoted” `<number-sign>` was treated as the start of a
 99221 comment. The `make` utility does not pay any attention to quotes. A `<number-sign>` starts a
 99222 comment regardless of its surroundings.

99223 The text about “other implementation-defined pathnames may also be tried” in addition to
 99224 `./makefile` and `./Makefile` is to allow such extensions as `SCCS/s.Makefile` and other variations.
 99225 It was made an implementation-defined requirement (as opposed to unspecified behavior) to
 99226 highlight surprising implementations that might select something unexpected like
 99227 `/etc/Makefile`. XSI-conformant systems also try `./s.makefile`, `SCCS/s.makefile`, `./s.Makefile`,
 99228 and `SCCS/s.Makefile`.

99229 Early proposals contained the macro `NPROC` as a means of specifying that `make` should use `n`
 99230 processes to do the work required. While this feature is a valuable extension for many systems, it
 99231 is not common usage and could require other non-trivial extensions to makefile syntax. This
 99232 extension is not required by this volume of POSIX.1-2017, but could be provided as a compatible
 99233 extension. The macro `PARALLEL` is used by some historical systems with essentially the same
 99234 meaning (but without using a name that is a common system limit value). It is suggested that
 99235 implementors recognize the existing use of `NPROC` and/or `PARALLEL` as extensions to `make`.

99236 The default rules are based on System V. The default `CC=` value is `c99` instead of `cc` because this
 99237 volume of POSIX.1-2017 does not standardize the utility named `cc`. Thus, every conforming
 99238 application would be required to define `CC=c99` to expect to run. There is no advantage
 99239 conferred by the hope that the makefile might hit the “preferred” compiler because this cannot
 99240 be guaranteed to work. Also, since the portable makescript can only use the `c99` options, no
 99241 advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue
 99242 as to whether `c99` is as valuable as `cc`.

99243 The `-d` option to `make` is frequently used to produce debugging information, but is too
 99244 implementation-defined to add to this volume of POSIX.1-2017.

99245 The `-p` option is not passed in `MAKEFLAGS` on most historical implementations and to change
 99246 this would cause many implementations to break without sufficiently increased portability.

99247 Commands that begin with a `<plus-sign>` (`'+'`) are executed even if the `-n` option is present.
 99248 Based on the GNU version of `make`, the behavior of `-n` when the `<plus-sign>` prefix is
 99249 encountered has been extended to apply to `-q` and `-t` as well. However, the System V
 99250 convention of forcing command execution with `-n` when the command line of a target contains
 99251 either of the strings `"$(MAKE)"` or `"${MAKE}"` has not been adopted. This functionality
 99252 appeared in early proposals, but the danger of this approach was pointed out with the following
 99253 example of a portion of a makefile:

```
99254 subdir:
99255     cd subdir; rm all_the_files; $(MAKE)
```

99256 The loss of the System V behavior in this case is well-balanced by the safety afforded to other
 99257 makefiles that were not aware of this situation. In any event, the command line `<plus-sign>`

99258 prefix can provide the desired functionality.

99259 The double <colon> in the target rule format is supported in BSD systems to allow more than
99260 one target line containing the same target name to have commands associated with it. Since this
99261 is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not
99262 mandated.

99263 The default rules are provided with text specifying that the built-in rules shall be the same as if
99264 the listed set were used. The intent is that implementations should be able to use the rules
99265 without change, but will be allowed to alter them in ways that do not affect the primary
99266 behavior.

99267 One point of discussion was whether to drop the default rules list from this volume of
99268 POSIX.1-2017. They provide convenience, but do not enhance portability of applications. The
99269 prime benefit is in portability of users who wish to type *make command* and have the command
99270 build from a **command.c** file.

99271 The historical *MAKESHELL* feature, and related features provided by other *make*
99272 implementations, were omitted. In some implementations it is used to let a user override the
99273 shell to be used to run *make* commands. This was confusing; for a portable *make*, the shell should
99274 be chosen by the makefile writer. Further, a makefile writer cannot require an alternate shell to
99275 be used and still consider the makefile portable. While it would be possible to standardize a
99276 mechanism for specifying an alternate shell, existing implementations do not agree on such a
99277 mechanism, and makefile writers can already invoke an alternate shell by specifying the shell
99278 name in the rule for a target; for example:

```
99279 python -c "foo"
```

99280 The *make* utilities in most historical implementations process the prerequisites of a target in left-
99281 to-right order, and the makefile format requires this. It supports the standard idiom used in
99282 many makefiles that produce *yacc* programs; for example:

```
99283 foo: y.tab.o lex.o main.o  
99284      $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

99285 In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct
99286 **y.tab.h**. Although there may be better ways to express this relationship, it is widely used
99287 historically. Implementations that desire to update prerequisites in parallel should require an
99288 explicit extension to *make* or the makefile format to accomplish it, as described previously.

99289 The algorithm for determining a new entry for target rules is partially unspecified. Some
99290 historical *makes* allow comment lines (including blank and empty lines) within the collection of
99291 commands marked by leading <tab> characters. A conforming makefile must ensure that each
99292 command starts with a <tab>, but implementations are free to ignore comments without
99293 triggering the start of a new entry.

99294 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with
99295 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do
99296 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned
99297 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it
99298 is required to resend itself the signal it received so that it exits with a status that reflects the
99299 signal. The results from SIGQUIT are partially unspecified because, on systems that create **core**
99300 files upon receipt of SIGQUIT, the **core** from *make* would conflict with a **core** file from the
99301 command that was running when the SIGQUIT arrived. The main concern was to prevent
99302 damaged files from appearing up-to-date when *make* is rerun.

99303 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no

99304 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;
 99305 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of
 99306 targets than for the entire makefile. These extensions to *make* in System V were made to match
 99307 historical practice from the BSD *make*.

99308 Macros are not exported to the environment of commands to be run. This was never the case in
 99309 any historical *make* and would have serious consequences. The environment is the same as the
 99310 environment to *make* except that **MAKEFLAGS** and macros defined on the *make* command line
 99311 are added, and except that macros defined by the **MAKEFLAGS** environment variable and
 99312 macros defined in the makefile(s) may update the value of an existing environment variable
 99313 (other than **SHELL**).

99314 Some implementations do not use *system()* for all command lines, as required by the portable
 99315 makefile format; as a performance enhancement, they select lines without shell metacharacters
 99316 for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but
 99317 merely that the same results be achieved. The metacharacters typically used to bypass the direct
 99318 *execve()* execution have been any of:

99319 = | ^ () ; & < > * ? [] : \$ ` ' " \ \n

99320 The default in some advanced versions of *make* is to group all the command lines for a target and
 99321 execute them using a single shell invocation; the System V method is to pass each line
 99322 individually to a separate shell. The single-shell method has the advantages in performance and
 99323 the lack of a requirement for many continued lines. However, converting to this newer method
 99324 has caused portability problems with many historical makefiles, so the behavior with the POSIX
 99325 makefile is specified to be the same as that of System V. It is suggested that the special target
 99326 **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a
 99327 target or group of targets.

99328 Novice users of *make* have had difficulty with the historical need to start commands with a
 99329 <tab>. Since it is often difficult to discern differences between <tab> and <space> characters on
 99330 terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made
 99331 to correct this problem by allowing leading <blank> characters instead of <tab> characters.
 99332 However, implementors reported many makefiles that failed in subtle ways following this
 99333 change, and it is difficult to implement a *make* that unambiguously can differentiate between
 99334 macro and command lines. There is extensive historical practice of allowing leading <space>
 99335 characters before macro definitions. Forcing macro lines into column 1 would be a significant
 99336 backwards-compatibility problem for some makefiles. Therefore, historical practice was
 99337 restored.

99338 There is substantial variation in the handling of include lines by different implementations.
 99339 However, there is enough commonality for the standard to be able to specify a minimum set of
 99340 requirements that allow the feature to be used portably. Known variations have been explicitly
 99341 called out as unspecified behavior in the description.

99342 The System V dynamic dependency feature was not included. It would support:

99343 cat: \$\$@.c

99344 that would expand to;

99345 cat: cat.c

99346 This feature exists only in the new version of System V *make* and, while useful, is not in wide
 99347 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time
 99348 and once at target update time.

99349 Consideration was given to adding metarules to the POSIX *make*. This would make **%.o: %.c** the

99350 same as **.c.o.**. This is quite useful and available from some vendors, but it would cause too many
99351 changes to this *make* to support. It would have introduced rule chaining and new substitution
99352 rules. However, the rules for target names have been set to reserve the '%' and '"' characters.
99353 These are traditionally used to implement metarules and quoting of target names, respectively.
99354 Implementors are strongly encouraged to use these characters only for these purposes.

99355 A request was made to extend the suffix delimiter character from a <period> to any character.
99356 The metarules feature in newer *makes* solves this problem in a more general way. This volume of
99357 POSIX.1-2017 is staying with the more conservative historical definition.

99358 The standard output format for the **-p** option is not described because it is primarily a
99359 debugging option and because the format is not generally useful to programs. In historical
99360 implementations the output is not suitable for use in generating makefiles. The **-p** format has
99361 been variable across historical implementations. Therefore, the definition of **-p** was only to
99362 provide a consistently named option for obtaining *make* script debugging information.

99363 Some historical implementations have not cleared the suffix list with **-r**.

99364 Implementations should be aware that some historical applications have intermixed *target_name*
99365 and *macro=value* operands on the command line, expecting that all of the macros are processed
99366 before any of the targets are dealt with. Conforming applications do not do this, but some
99367 backwards-compatibility support may be warranted.

99368 Empty inference rules are specified with a <semicolon> command rather than omitting all
99369 commands, as described in an early proposal. The latter case has no traditional meaning and is
99370 reserved for implementation extensions, such as in GNU *make*.

99371 Earlier versions of this standard defined comment lines only as lines with '#' as the first
99372 character. Many places then talked about comments, blank lines, and empty lines; but some
99373 places inadvertently only mentioned comments when blank lines and empty lines had also been
99374 accepted in all known implementations. The standard now defines comment lines to be blank
99375 lines, empty lines, and lines starting with a '#' character and explicitly lists cases where blank
99376 lines and empty lines are not acceptable.

99377 On most historic systems, the *make* utility considered a target with a prerequisite that had an
99378 identical timestamp as up-to-date. The HP-UX implementation of *make* treated it as out-of-date.
99379 The standard now allows either behavior, but implementations are encouraged to follow the
99380 example set by HP-UX. This is especially important on file systems where the timestamp
99381 resolution is the minimum (1 second) required by the standard. All implementations of *make*
99382 should make full use of the finest timestamp resolution available on the file systems holding
99383 targets and prerequisites to ensure that targets are up-to-date even for prerequisite files with
99384 timestamps that were updated within the same second. However, if the timestamp resolutions of
99385 the file systems containing a target and a prerequisite are different, the timestamp with the more
99386 precise resolution should be rounded down to the resolution of the less precise timestamp for
99387 the comparison.

99388 FUTURE DIRECTIONS

99389 Some implementations of *make* include an *export* directive to add specified *make* variables to the
99390 environment. This may be considered for standardization in a future version.

99391 A future version of this standard may require that macro expansions using the forms
99392 $\$(string1:[op]@[os]=[np][%][ns])$ or $\${string1:[op]@[os]=[np][%][ns]}$ are treated as pattern macro
99393 expansions.

99394 **SEE ALSO**

- 99395 [Chapter 2](#) (on page 2345), [ar](#), [c99](#), [get](#), [lex](#), [sccs](#), [sh](#), [yacc](#)
- 99396 [XBD Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 99397 [XSH *exec*, *system\(\)*](#)

99398 **CHANGE HISTORY**

- 99399 First released in Issue 2.

99400 **Issue 5**

- 99401 The FUTURE DIRECTIONS section is added.

99402 **Issue 6**

- 99403 This utility is marked as part of the Software Development Utilities option.
- 99404 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the SPECIAL TARGETS section.
- 99406 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.
- 99407
- 99408
- 99409 It is specified whether the command line is related to the makefile or to the *make* command, and the macro processing rules are updated to align with the IEEE P1003.2b draft standard.
- 99410
- 99411 The normative text is reworded to avoid use of the term “must” for application requirements.
- 99412 PASC Interpretation 1003.2 #193 is applied.

99413 **Issue 7**

- 99414 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.
- 99415
- 99416 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 99417 Include lines in makefiles are introduced.
- 99418 Austin Group Interpretation 1003.1-2001 #131 is applied, changing the **Makefile Execution** section.
- 99419
- 99420 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0121 [257] is applied.
- 99421 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0122 [509], XCU/TC2-2008/0123 [584], XCU/TC2-2008/0124 [857], XCU/TC2-2008/0125 [505], XCU/TC2-2008/0126 [584], XCU/TC2-2008/0127 [505], XCU/TC2-2008/0128 [865], XCU/TC2-2008/0129 [693], XCU/TC2-2008/0130 [602], XCU/TC2-2008/0131 [848], XCU/TC2-2008/0132 [763], XCU/TC2-2008/0133 [857], XCU/TC2-2008/0134 [866], XCU/TC2-2008/0135 [525], XCU/TC2-2008/0136 [848], XCU/TC2-2008/0137 [769], XCU/TC2-2008/0138 [525], XCU/TC2-2008/0139 [769], XCU/TC2-2008/0140 [505], XCU/TC2-2008/0141 [693], XCU/TC2-2008/0142 [505], XCU/TC2-2008/0143 [857], and XCU/TC2-2008/0144 [693,865] are applied.
- 99428
- 99429

99430 **NAME**

99431 man ‡display system documentation

99432 **SYNOPSIS**99433 man [-k] *name*...99434 **DESCRIPTION**

99435 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a
 99436 standard utility, *man* at a minimum shall write a message describing the syntax used by the
 99437 standard utility, its options, and operands. If more information is available, the *man* utility shall
 99438 provide it in an implementation-defined manner.

99439 An implementation may provide information for values of *name* other than the standard utilities.
 99440 Standard utilities that are listed as optional and that are not supported by the implementation
 99441 either shall cause a brief message indicating that fact to be displayed or shall cause a full display
 99442 of information as described previously.

99443 **OPTIONS**99444 The *man* utility shall conform to XBD [Section 12.2](#) (on page 216).

99445 The following option shall be supported:

99446 **-k** Interpret *name* operands as keywords to be used in searching a utilities summary
 99447 database that contains a brief purpose entry for each standard utility and write lines
 99448 from the summary database that match any of the keywords. The keyword search shall
 99449 produce results that are the equivalent of the output of the following command:

```
99450 grep -Ei '
99451     name
99452     name
99453     ...
99454     ' summary-database
```

99455 This assumes that the *summary-database* is a text file with a single entry per line; this
 99456 organization is not required and the example using *grep -Ei* is merely illustrative of the
 99457 type of search intended. The purpose entry to be included in the database shall consist
 99458 of a terse description of the purpose of the utility.

99459 **OPERANDS**

99460 The following operand shall be supported:

99461 *name* A keyword or the name of a standard utility. When **-k** is not specified and *name*
 99462 does not represent one of the standard utilities, the results are unspecified.

99463 **STDIN**

99464 Not used.

99465 **INPUT FILES**

99466 None.

99467 **ENVIRONMENT VARIABLES**99468 The following environment variables shall affect the execution of *man*:

99469 **LANG** Provide a default value for the internationalization variables that are unset or null.
 99470 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 99471 variables used to determine the values of locale categories.)

99472 **LC_ALL** If set to a non-empty string value, override the values of all the other
 99473 internationalization variables.

99474 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 99475 characters (for example, single-byte as opposed to multi-byte characters in
 99476 arguments and in the summary database). The value of *LC_CTYPE* need not affect
 99477 the format of the information written about the *name* operands.

99478 **LC_MESSAGES**
 99479 Determine the locale that should be used to affect the format and contents of
 99480 diagnostic messages written to standard error and informative messages written to
 99481 standard output.

99482 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

99483 **PAGER** Determine an output filtering command for writing the output to a terminal. Any
 99484 string acceptable as a *command_string* operand to the *sh -c* command shall be valid.
 99485 When standard output is a terminal device, the reference page output shall be
 99486 piped through the command. If the *PAGER* variable is null or not set, the command
 99487 shall be either *more* or another paginator utility documented in the system
 99488 documentation.

99489 **ASYNCHRONOUS EVENTS**
 99490 Default.

99491 **STDOUT**
 99492 The *man* utility shall write text describing the syntax of the utility *name*, its options and its
 99493 operands, or, when *-k* is specified, lines from the summary database. The format of this text is
 99494 implementation-defined.

99495 **STDERR**
 99496 The standard error shall be used for diagnostic messages, and may also be used for
 99497 informational messages of unspecified format.

99498 **OUTPUT FILES**
 99499 None.

99500 **EXTENDED DESCRIPTION**
 99501 None.

99502 **EXIT STATUS**
 99503 The following exit values shall be returned:
 99504 0 Successful completion.
 99505 >0 An error occurred.

99506 **CONSEQUENCES OF ERRORS**
 99507 Default.

99508 **APPLICATION USAGE**
 99509 None.

99510 **EXAMPLES**
 99511 None.

99512 **RATIONALE**
 99513 It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the
 99514 standard developers was strongly divided as to how much or how little information *man* should
 99515 be required to provide. They considered, however, that the provision of some portable way of
 99516 accessing documentation would aid user portability. The arguments against a fuller specification
 99517 were:

99518 Large quantities of documentation should not be required on a system that does not have
99519 excess disk space.

99520 The current manual system does not present information in a manner that greatly aids user
99521 portability.

99522 A “better help system” is currently an area in which vendors feel that they can add value
99523 to their POSIX implementations.

99524 The `-f` option was considered, but due to implementation differences, it was not included in this
99525 volume of POSIX.1-2017.

99526 The description was changed to be more specific about what has to be displayed for a utility. The
99527 standard developers considered it insufficient to allow a display of only the synopsis without
99528 giving a short description of what each option and operand does.

99529 The “purpose” entry to be included in the database can be similar to the section title (less the
99530 numeric prefix) from this volume of POSIX.1-2017 for each utility. These titles are similar to
99531 those used in historical systems for this purpose.

99532 See *mailx* for rationale concerning the default paginator.

99533 The caveat in the *LC_CTYPE* description was added because it is not a requirement that an
99534 implementation provide reference pages for all of its supported locales on each system;
99535 changing *LC_CTYPE* does not necessarily translate the reference page into another language.
99536 This is equivalent to the current state of *LC_MESSAGES* in POSIX.1-2017 ‡locale-specific
99537 messages are not yet a requirement.

99538 The historical *MANPATH* variable is not included in POSIX because no attempt is made to
99539 specify naming conventions for reference page files, nor even to mandate that they are files at
99540 all. On some implementations they could be a true database, a hypertext file, or even fixed
99541 strings within the *man* executable. The standard developers considered the portability of
99542 reference pages to be outside their scope of work. However, users should be aware that
99543 *MANPATH* is implemented on a number of historical systems and that it can be used to tailor
99544 the search pattern for reference pages from the various categories (utilities, functions, file
99545 formats, and so on) when the system administrator reveals the location and conventions for
99546 reference pages on the system.

99547 The keyword search can rely on at least the text of the section titles from these utility
99548 descriptions, and the implementation may add more keywords. The term “section titles” refers
99549 to the strings such as:

99550 `man` – Display system documentation
99551 `ps` – Report process status

99552 FUTURE DIRECTIONS

99553 None.

99554 SEE ALSO

99555 [more](#)

99556 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

99557 CHANGE HISTORY

99558 First released in Issue 4.

99559 **Issue 5**

99560 The FUTURE DIRECTIONS section is added.

99561 **Issue 7**

99562 Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages
99563 may appear on standard error.

99564 **NAME**

99565 mesg ‡permit or deny messages

99566 **SYNOPSIS**

99567 mesg [y|n]

99568 **DESCRIPTION**

99569 The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or
 99570 other utilities to a terminal device. The terminal device affected shall be determined by searching
 99571 for the first terminal in the sequence of devices associated with standard input, standard output,
 99572 and standard error, respectively. With no arguments, *mesg* shall report the current state without
 99573 changing it. Processes with appropriate privileges may be able to send messages to the terminal
 99574 independent of the current state.

99575 **OPTIONS**

99576 None.

99577 **OPERANDS**

99578 The following operands shall be supported in the POSIX locale:

99579 *y* Grant permission to other users to send messages to the terminal device.99580 *n* Deny permission to other users to send messages to the terminal device.99581 **STDIN**

99582 Not used.

99583 **INPUT FILES**

99584 None.

99585 **ENVIRONMENT VARIABLES**99586 The following environment variables shall affect the execution of *mesg*:

99587 *LANG* Provide a default value for the internationalization variables that are unset or null.
 99588 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 99589 variables used to determine the values of locale categories.)

99590 *LC_ALL* If set to a non-empty string value, override the values of all the other
 99591 internationalization variables.

99592 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 99593 characters (for example, single-byte as opposed to multi-byte characters in
 99594 arguments).

99595 *LC_MESSAGES*

99596 Determine the locale that should be used to affect the format and contents of
 99597 diagnostic messages written (by *mesg*) to standard error.

99598 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.99599 **ASYNCHRONOUS EVENTS**

99600 Default.

99601 **STDOUT**99602 If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.99603 **STDERR**

99604 The standard error shall be used only for diagnostic messages.

99605 OUTPUT FILES

99606 None.

99607 EXTENDED DESCRIPTION

99608 None.

99609 EXIT STATUS

99610 The following exit values shall be returned:

99611 0 Receiving messages is allowed.

99612 1 Receiving messages is not allowed.

99613 >1 An error occurred.

99614 CONSEQUENCES OF ERRORS

99615 Default.

99616 APPLICATION USAGE

99617 The mechanism by which the message status of the terminal is changed is unspecified.
99618 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has
99619 successfully completed. These actions may include, but are not limited to: another invocation of
99620 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or
99621 *chmod()* function, and so on.

99622 EXAMPLES

99623 None.

99624 RATIONALE

99625 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather
99626 than the controlling terminal for the session. This is because users logged in more than once
99627 should be able to change any of their login terminals without having to stop the job running in
99628 those sessions. This is not a security problem involving the terminals of other users because
99629 appropriate privileges would be required to affect the terminal of another user.

99630 The method of checking each of the first three file descriptors in sequence until a terminal is
99631 found was adopted from System V.

99632 The file */dev/tty* is not specified for the terminal device because it was thought to be too
99633 restrictive. Typical environment changes for the *n* operand are that write permissions are
99634 removed for *others* and *group* from the appropriate device. It was decided to leave the actual
99635 description of what is done as unspecified because of potential differences between
99636 implementations.

99637 The format for standard output is unspecified because of differences between historical
99638 implementations. This output is generally not useful to shell scripts (they can use the exit status),
99639 so exact parsing of the output is unnecessary.

99640 FUTURE DIRECTIONS

99641 None.

99642 SEE ALSO

99643 *talk*, *write*

99644 XBD Chapter 8 (on page 173)

99645 **CHANGE HISTORY**

99646 First released in Issue 2.

99647 **Issue 6**

99648 This utility is marked as part of the User Portability Utilities option.

99649 **Issue 7**99650 The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability
99651 Utilities is now an option for interactive utilities.

99652 **NAME**

99653 mkdir — make directories

99654 **SYNOPSIS**99655 mkdir [-p] [-m *mode*] *dir*...99656 **DESCRIPTION**99657 The *mkdir* utility shall create the directories specified by the operands, in the order specified.99658 For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function
99659 defined in the System Interfaces volume of POSIX.1-2017, called with the following arguments:

- 99660 1. The *dir* operand is used as the *path* argument.
- 99661 2. The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as
99662 the *mode* argument. (If the **-m** option is specified, the value of the *mkdir()* *mode* argument
99663 is unspecified, but the directory shall at no time have permissions less restrictive than the
99664 **-m mode** option-argument.)

99665 **OPTIONS**99666 The *mkdir* utility shall conform to XBD [Section 12.2](#) (on page 216).

99667 The following options shall be supported:

99668 **-m mode** Set the file permission bits of the newly-created directory to the specified *mode*
99669 value. The *mode* option-argument shall be the same as the *mode* operand defined
99670 for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-'
99671 shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add
99672 permissions to the default mode, '-' shall delete permissions from the default
99673 mode.

99674 **-p** Create any missing intermediate pathname components.

99675 For each *dir* operand that does not name an existing directory, before performing
99676 the actions described in the DESCRIPTION above, the *mkdir* utility shall create any
99677 pathname components of the path prefix of *dir* that do not name an existing
99678 directory by performing actions equivalent to first calling the *mkdir()* function with
99679 the following arguments:

- 99680 1. A pathname naming the missing pathname component, ending with a
99681 trailing <slash> character, as the *path* argument
- 99682 2. The value zero as the *mode* argument

99683 and then calling the *chmod()* function with the following arguments:

- 99684 1. The same *path* argument as in the *mkdir()* call
- 99685 2. The value $(S_IWUSR|S_IXUSR|\sim filemask)\&0777$ as the *mode*
99686 argument, where *filemask* is the file mode creation mask of the process (see
99687 XSH *umask()*)

99688 Each *dir* operand that names an existing directory shall be ignored without error.

99689 **OPERANDS**

99690 The following operand shall be supported:

99691 *dir* A pathname of a directory to be created.

99692 **STDIN**
 99693 Not used.

99694 **INPUT FILES**
 99695 None.

99696 **ENVIRONMENT VARIABLES**
 99697 The following environment variables shall affect the execution of *mkdir*:

99698 *LANG* Provide a default value for the internationalization variables that are unset or null.
 99699 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 99700 variables used to determine the values of locale categories.)

99701 *LC_ALL* If set to a non-empty string value, override the values of all the other
 99702 internationalization variables.

99703 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 99704 characters (for example, single-byte as opposed to multi-byte characters in
 99705 arguments).

99706 *LC_MESSAGES*
 99707 Determine the locale that should be used to affect the format and contents of
 99708 diagnostic messages written to standard error.

99709 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

99710 **ASYNCHRONOUS EVENTS**
 99711 Default.

99712 **STDOUT**
 99713 Not used.

99714 **STDERR**
 99715 The standard error shall be used only for diagnostic messages.

99716 **OUTPUT FILES**
 99717 None.

99718 **EXTENDED DESCRIPTION**
 99719 None.

99720 **EXIT STATUS**
 99721 The following exit values shall be returned:

99722 0 All the specified directories were created successfully, or the **-p** option was specified and all
 99723 the specified directories either already existed or were created successfully.

99724 >0 An error occurred.

99725 **CONSEQUENCES OF ERRORS**
 99726 Default.

99727 APPLICATION USAGE

99728 The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions
99729 removed in accordance with the file mode creation mask. For intermediate pathname
99730 components created by *mkdir*, the mode is the default modified by *u+rwx* so that the
99731 subdirectories can always be created regardless of the file mode creation mask; if different
99732 ultimate permissions are desired for the intermediate directories, they can be changed
99733 afterwards with *chmod*.

99734 Note that some of the requested directories may have been created even if an error occurs.

99735 EXAMPLES

99736 None.

99737 RATIONALE

99738 The System V *-m* option was included to control the file mode.

99739 The System V *-p* option was included to create any needed intermediate directories and to
99740 complement the functionality provided by *rmdir* for removing directories in the path prefix as
99741 they become empty. Because no error is produced if any path component already exists, the *-p*
99742 option is also useful to ensure that a particular directory exists.

99743 The functionality of *mkdir* is described substantially through a reference to the *mkdir()* function
99744 in the System Interfaces volume of POSIX.1-2017. For example, by default, the mode of the
99745 directory is affected by the file mode creation mask in accordance with the specified behavior of
99746 the *mkdir()* function. In this way, there is less duplication of effort required for describing details
99747 of the directory creation.

99748 FUTURE DIRECTIONS

99749 None.

99750 SEE ALSO

99751 *chmod*, *rm*, *rmdir*, *umask*

99752 XBD Chapter 8 (on page 173), Section 12.2 (on page 216)

99753 XSH *mkdir()*, *umask()*

99754 CHANGE HISTORY

99755 First released in Issue 2.

99756 Issue 5

99757 The FUTURE DIRECTIONS section is added.

99758 Issue 7

99759 SD5-XCU-ERN-56 is applied, aligning the *-m* option with the IEEE P1003.2b draft standard to
99760 clarify an ambiguity.

99761 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

99762 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0122 [161] is applied.

99763 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0145 [843] is applied.

99764 **NAME**

99765 mkfifo ‡make FIFO special files

99766 **SYNOPSIS**99767 mkfifo [-m *mode*] *file*...99768 **DESCRIPTION**99769 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order
99770 specified.99771 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo()* function
99772 defined in the System Interfaces volume of POSIX.1-2017, called with the following arguments:

- 99773 1. The *file* operand is used as the *path* argument.
- 99774 2. The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP,
99775 S_IROTH, and S_IWOTH is used as the *mode* argument. (If the **-m** option is specified, the
99776 value of the *mkfifo()* *mode* argument is unspecified, but the FIFO shall at no time have
99777 permissions less restrictive than the **-m mode** option-argument.)

99778 **OPTIONS**99779 The *mkfifo* utility shall conform to XBD [Section 12.2](#) (on page 216).

99780 The following option shall be supported:

99781 **-m mode** Set the file permission bits of the newly-created FIFO to the specified *mode* value.
99782 The *mode* option-argument shall be the same as the *mode* operand defined for the
99783 *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-' shall be
99784 interpreted relative to an assumed initial mode of *a=rw*.

99785 **OPERANDS**

99786 The following operand shall be supported:

99787 *file* A pathname of the FIFO special file to be created.99788 **STDIN**

99789 Not used.

99790 **INPUT FILES**

99791 None.

99792 **ENVIRONMENT VARIABLES**99793 The following environment variables shall affect the execution of *mkfifo*:

99794 **LANG** Provide a default value for the internationalization variables that are unset or null.
99795 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
99796 variables used to determine the values of locale categories.)

99797 **LC_ALL** If set to a non-empty string value, override the values of all the other
99798 internationalization variables.

99799 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
99800 characters (for example, single-byte as opposed to multi-byte characters in
99801 arguments).

99802 **LC_MESSAGES**

99803 Determine the locale that should be used to affect the format and contents of
99804 diagnostic messages written to standard error.

- 99805 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 99806 **ASYNCHRONOUS EVENTS**
- 99807 Default.
- 99808 **STDOUT**
- 99809 Not used.
- 99810 **STDERR**
- 99811 The standard error shall be used only for diagnostic messages.
- 99812 **OUTPUT FILES**
- 99813 None.
- 99814 **EXTENDED DESCRIPTION**
- 99815 None.
- 99816 **EXIT STATUS**
- 99817 The following exit values shall be returned:
- 99818 0 All the specified FIFO special files were created successfully.
- 99819 >0 An error occurred.
- 99820 **CONSEQUENCES OF ERRORS**
- 99821 Default.
- 99822 **APPLICATION USAGE**
- 99823 None.
- 99824 **EXAMPLES**
- 99825 None.
- 99826 **RATIONALE**
- 99827 This utility was added to permit shell applications to create FIFO special files.
- 99828 The **-m** option was added to control the file mode, for consistency with the similar functionality provided by the *mkdir* utility.
- 99829
- 99830 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.
- 99831
- 99832
- 99833 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function in the System Interfaces volume of POSIX.1-2017. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of the *mkfifo()* function. In this way, there is less duplication of effort required for describing details of the file creation.
- 99834
- 99835
- 99836
- 99837
- 99838 **FUTURE DIRECTIONS**
- 99839 None.
- 99840 **SEE ALSO**
- 99841 *chmod*, *umask*
- 99842 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 99843 XSH *mkfifo()*

99844 **CHANGE HISTORY**

99845 First released in Issue 3.

99846 **Issue 6**

99847 The `-m` option is aligned with the IEEE P1003.2b draft standard to clarify an ambiguity.

99848 **NAME**99849 `more` *filter* display files on a page-by-page basis99850 **SYNOPSIS**99851 UP `more [-ceisu] [-n number] [-p command] [-t tagstring] [file...]`99852 **DESCRIPTION**

99853 The *more* utility shall read files and either write them to the terminal on a page-by-page basis or
 99854 filter them to standard output. If standard output is not a terminal device, all input files shall be
 99855 copied to standard output in their entirety, without modification, except as specified for the `-s`
 99856 option. If standard output is a terminal device, the files shall be written a number of lines (one
 99857 screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION
 99858 section.

99859 Certain block-mode terminals do not have all the capabilities necessary to support the complete
 99860 *more* definition; they are incapable of accepting commands that are not terminated with a
 99861 `<newline>`. Implementations that support such terminals shall provide an operating mode to
 99862 *more* in which all commands can be terminated with a `<newline>` on those terminals. This mode:

99863 Shall be documented in the system documentation

99864 Shall, at invocation, inform the user of the terminal deficiency that requires the `<newline>`
 99865 usage and provide instructions on how this warning can be suppressed in future
 99866 invocations

99867 Shall not be required for implementations supporting only fully capable terminals

99868 Shall not affect commands already requiring `<newline>` characters

99869 Shall not affect users on the capable terminals from using *more* as described in this volume
 99870 of POSIX.1-2017

99871 **OPTIONS**

99872 The *more* utility shall conform to XBD [Section 12.2](#) (on page 216), except that '+' may be
 99873 recognized as an option delimiter as well as '-'.
 99874 The following options shall be supported:

99875 `-c` If a screen is to be written that has no lines in common with the current screen, or
 99876 *more* is writing its first screen, *more* shall not scroll the screen, but instead shall
 99877 redraw each line of the screen in turn, from the top of the screen to the bottom. In
 99878 addition, if *more* is writing its first screen, the screen shall be cleared. This option
 99879 may be silently ignored on devices with insufficient terminal capabilities.

99880 `-e` Exit immediately after writing the last line of the last file in the argument list; see
 99881 the EXTENDED DESCRIPTION section.

99882 `-i` Perform pattern matching in searches without regard to case; see XBD [Section 9.2](#)
 99883 (on page 182).

99884 `-n number` Specify the number of lines per screenful. The *number* argument is a positive
 99885 decimal integer. The `-n` option shall override any values obtained from any other
 99886 source.

99887 `-p command` Each time a screen from a new file is displayed or redisplayed (including as a
 99888 result of *more* commands; for example, `:p`), execute the *more* command(s) in the
 99889 command arguments in the order specified, as if entered by the user after the first
 99890 screen has been displayed. No intermediate results shall be displayed (that is, if the
 99891 command is a movement to a screen different from the normal first screen, only the

- 99892 screen resulting from the command shall be displayed.) If any of the commands
 99893 fail for any reason, an informational message to this effect shall be written, and no
 99894 further commands specified using the **-p** option shall be executed for this file.
- 99895 **-s** Behave as if consecutive empty lines were a single empty line.
- 99896 **-t tagstring** Write the screenful of the file containing the tag named by the *tagstring*
 99897 argument. See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**
 99898 command is optional. It shall be provided on any system that also provides a
 99899 conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined
 99900 results.
- 99901 The filename resulting from the **-t** option shall be logically added as a prefix to the
 99902 list of command line files, as if specified by the user. If the tag named by the
 99903 *tagstring* argument is not found, it shall be an error, and *more* shall take no further
 99904 action.
- 99905 If the tag specifies a line number, the first line of the display shall contain the
 99906 beginning of that line. If the tag specifies a pattern, the first line of the display shall
 99907 contain the beginning of the matching text from the first line of the file that
 99908 contains that pattern. If the line does not exist in the file or matching text is not
 99909 found, an informational message to this effect shall be displayed, and *more* shall
 99910 display the default screen as if **-t** had not been specified.
- 99911 If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be
 99912 processed first; that is, the file and starting line for the display shall be as specified
 99913 by **-t**, and then the **-p more** command shall be executed. If the line (matching text)
 99914 specified by the **-t** command does not exist (is not found), no **-p more** command
 99915 shall be executed for this file at any time.
- 99916 **-u** Treat a <backspace> as a printable control character, displayed as an
 99917 implementation-defined character sequence (see the EXTENDED DESCRIPTION
 99918 section), suppressing backspacing and the special handling that produces
 99919 underlined or standout mode text on some terminal types. Also, do not ignore a
 99920 <carriage-return> at the end of a line.

OPERANDS

99921 The following operand shall be supported:

- 99923 *file* A pathname of an input file. If no *file* operands are specified, the standard input
 99924 shall be used. If a *file* is '-', the standard input shall be read at that point in the
 99925 sequence.

STDIN

99926 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

INPUT FILES

99928 The input files being examined shall be text files. If standard output is a terminal, standard error
 99929 shall be used to read commands from the user. If standard output is a terminal, standard error is
 99930 not readable, and command input is needed, *more* may attempt to obtain user commands from
 99931 the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error
 99932 indicating that it was unable to read user commands. If standard output is not a terminal, no
 99933 error shall result if standard error cannot be opened for reading.

99935 **ENVIRONMENT VARIABLES**99936 The following environment variables shall affect the execution of *more*:99937 *COLUMNS* Override the system-selected horizontal display line size. See XBD [Chapter 8](#) (on
99938 page 173) for valid values and results when it is unset or null.99939 *EDITOR* Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION
99940 section.99941 *LANG* Provide a default value for the internationalization variables that are unset or null.
99942 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
99943 variables used to determine the values of locale categories.)99944 *LC_ALL* If set to a non-empty string value, override the values of all the other
99945 internationalization variables.99946 *LC_COLLATE*99947 Determine the locale for the behavior of ranges, equivalence classes, and multi-
99948 character collating elements within regular expressions.99949 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
99950 characters (for example, single-byte as opposed to multi-byte characters in
99951 arguments and input files) and the behavior of character classes within regular
99952 expressions.99953 *LC_MESSAGES*99954 Determine the locale that should be used to affect the format and contents of
99955 diagnostic messages written to standard error and informative messages written to
99956 standard output.99957 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.99958 *LINES* Override the system-selected vertical screen size, used as the number of lines in a
99959 screenful. See XBD [Chapter 8](#) (on page 173) for valid values and results when it is
99960 unset or null. The *-n* option shall take precedence over the *LINES* variable for
99961 determining the number of lines in a screenful.99962 *MORE* Determine a string containing options described in the OPTIONS section preceded
99963 with <hyphen-minus> characters and <blank>-separated as on the command line.
99964 Any command line options shall be processed after those in the *MORE* variable, as
99965 if the command line were:99966 *more* \$MORE *options operands*99967 The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for
99968 determining the number of lines in a screenful.99969 *TERM* Determine the name of the terminal type. If this variable is unset or null, an
99970 unspecified default terminal type is used.99971 **ASYNCHRONOUS EVENTS**

99972 Default.

99973 **STDOUT**

99974 The standard output shall be used to write the contents of the input files.

99975 **STDERR**

99976 The standard error shall be used for diagnostic messages and user commands (see the INPUT
 99977 FILES section), and, if standard output is a terminal device, to write a prompting string. The
 99978 prompting string shall appear on the screen line below the last line of the file displayed in the
 99979 current screenful. The prompt shall contain the name of the file currently being examined and
 99980 shall contain an end-of-file indication and the name of the next file, if any, when prompting at
 99981 the end-of-file. If an error or informational message is displayed, it is unspecified whether it is
 99982 contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the
 99983 user shall be prompted for a continuation character, at which point another message or the user
 99984 prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether
 99985 informational messages are written for other user commands.

99986 **OUTPUT FILES**

99987 None.

99988 **EXTENDED DESCRIPTION**

99989 The following section describes the behavior of *more* when the standard output is a terminal
 99990 device. If the standard output is not a terminal device, no options other than `-s` shall have any
 99991 effect, and all input files shall be copied to standard output otherwise unmodified, at which time
 99992 *more* shall exit without further action.

99993 The number of lines available per screen shall be determined by the `-n` option, if present, or by
 99994 examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither
 99995 method yields a number, an unspecified number of lines shall be used.

99996 The maximum number of lines written shall be one less than this number, because the screen
 99997 line after the last line written shall be used to write a user prompt and user input. If the number
 99998 of lines in the screen is less than two, the results are undefined. It is unspecified whether user
 99999 input is permitted to be longer than the remainder of the single line where the prompt has been
 100000 written.

100001 The number of columns available per line shall be determined by examining values in the
 100002 environment (see the ENVIRONMENT VARIABLES section), with a default value as described
 100003 in XBD [Chapter 8](#) (on page 173).

100004 Lines that are longer than the display shall be folded; the length at which folding occurs is
 100005 unspecified, but should be appropriate for the output device. Folding may occur between glyphs
 100006 of single characters that take up multiple display columns.

100007 When standard output is a terminal and `-u` is not specified, *more* shall treat `<backspace>` and
 100008 `<carriage-return>` characters specially:

100009 A character, followed first by a sequence of *n* `<backspace>` characters (where *n* is the same
 100010 as the number of column positions that the character occupies), then by *n* `<underscore>`
 100011 characters (`'_'`), shall cause that character to be written as underlined text, if the terminal
 100012 type supports that. The *n* `<underscore>` characters, followed first by *n* `<backspace>`
 100013 characters, then any character with *n* column positions, shall also cause that character to be
 100014 written as underlined text, if the terminal type supports that.

100015 A sequence of *n* `<backspace>` characters (where *n* is the same as the number of column
 100016 positions that the previous character occupies) that appears between two identical
 100017 printable characters shall cause the first of those two characters to be written as
 100018 emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the
 100019 terminal type supports that, and the second to be discarded. Immediately subsequent
 100020 occurrences of `<backspace>/character` pairs for that same character shall also be
 100021 discarded. (For example, the sequence `"a\ba\ba\ba"` is interpreted as a single

100022 emboldened 'a'.)

100023 The *more* utility shall logically discard all other <backspace> characters from the line as
100024 well as the character which precedes them, if any.

100025 A <carriage-return> at the end of a line shall be ignored, rather than being written as a
100026 non-printable character, as described in the next paragraph.

100027 It is implementation-defined how other non-printable characters are written. Implementations
100028 should use the same format that they use for the *ex print* command; see the OPTIONS section
100029 within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it
100030 crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the
100031 number of columns on the display is less than the number of columns any single character in the
100032 line being displayed would occupy.

100033 When each new file is displayed (or redisplayed), *more* shall write the first screen of the file.
100034 Once the initial screen has been written, *more* shall prompt for a user command. If the execution
100035 of the user command results in a screen that has lines in common with the current screen, and
100036 the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is
100037 unspecified whether the screen is scrolled or redrawn.

100038 For all files but the last (including standard input if no file was specified, and for the last file as
100039 well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall
100040 prompt for a user command. This prompt shall contain the name of the next file as well as an
100041 indication that *more* has reached end-of-file. If the user command is *f*, <control>-F, <space>, *j*,
100042 <newline>, *d*, <control>-D, or *s*, *more* shall display the next file. Otherwise, if displaying the last
100043 file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

100044 Several of the commands described in this section display a previous screen from the input
100045 stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is
100046 implementation-defined how much backwards motion is supported. If a command cannot be
100047 executed because of a limitation on backwards motion, an error message to this effect shall be
100048 displayed, the current screen shall not change, and the user shall be prompted for another
100049 command.

100050 If a command cannot be performed because there are insufficient lines to display, *more* shall alert
100051 the terminal. If a command cannot be performed because there are insufficient lines to display or
100052 a */* command fails: if the input is the standard input, the last screen in the file may be displayed;
100053 otherwise, the current file and screen shall not change, and the user shall be prompted for
100054 another command.

100055 The interactive commands in the following sections shall be supported. Some commands can be
100056 preceded by a decimal integer, called *count* in the following descriptions. If not specified with
100057 the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular
100058 expression, as described in XBD [Section 9.3](#) (on page 183). The term “examine” is historical
100059 usage meaning “open the file for viewing”; for example, *more foo* would be expressed as
100060 examining file **foo**.

100061 In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a
100062 line from the file being examined.

100063 In the following descriptions, the *current position* refers to two things:

100064 1. The position of the current line on the screen

100065 2. The line number (in the file) of the current line on the screen

100066 Usually, the line on the screen corresponding to the current position is the third line on the

100067 screen. If this is not possible (there are fewer than three lines to display or this is the first page of
 100068 the file, or it is the last page of the file), then the current position is either the first or last line on
 100069 the screen as described later.

100070 Help

100071 *Synopsis:* h

100072 Write a summary of these commands and other implementation-defined commands. The
 100073 behavior shall be as if the *more* utility were executed with the `-e` option on a file that contained
 100074 the summary information. The user shall be prompted as described earlier in this section when
 100075 end-of-file is reached. If the user command is one of those specified to continue to the next file,
 100076 *more* shall return to the file and screen state from which the `h` command was executed.

100077 Scroll Forward One Screenful

100078 *Synopsis:* [*count*]f
 100079 [*count*]<control>-F

100080 Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size,
 100081 only the final screenful shall be written.

100082 Scroll Backward One Screenful

100083 *Synopsis:* [*count*]b
 100084 [*count*]<control>-B

100085 Scroll backward *count* lines, with a default of one screenful (see the `-n` option). If *count* is more
 100086 than the screen size, only the final screenful shall be written.

100087 Scroll Forward One Line

100088 *Synopsis:* [*count*]<space>
 100089 [*count*]j
 100090 [*count*]<newline>

100091 Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for `j` and
 100092 <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen
 100093 size.

100094 Scroll Backward One Line

100095 *Synopsis:* [*count*]k

100096 Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the
 100097 screen size.

100098 Scroll Forward One Half Screenful

100099 *Synopsis:* [*count*]d
 100100 [*count*]<control>-D

100101 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it
 100102 shall become the new default for subsequent `d`, <control>-D, and `u` commands.

100103 Skip Forward One Line

100104 *Synopsis:* [count]s

100105 Display the screenful beginning with the line *count* lines after the last line on the current screen.
100106 If *count* would cause the current position to be such that less than one screenful would be
100107 written, the last screenful in the file shall be written.

100108 Scroll Backward One Half Screenful

100109 *Synopsis:* [count]u
100110 [count]<control>-U

100111 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it
100112 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.
100113 The entire *count* lines shall be written, even if *count* is more than the screen size.

100114 Go to Beginning of File

100115 *Synopsis:* [count]g

100116 Display the screenful beginning with line *count*.

100117 Go to End-of-File

100118 *Synopsis:* [count]G

100119 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the
100120 last screenful of the file.

100121 Refresh the Screen

100122 *Synopsis:* r
100123 <control>-L

100124 Refresh the screen.

100125 Discard and Refresh

100126 *Synopsis:* R

100127 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered
100128 input shall not be discarded and the **R** command shall be equivalent to the **r** command.

100129 Mark Position

100130 *Synopsis:* m*letter*

100131 Mark the current position with the letter named by *letter*, where *letter* represents the name of one
100132 of the lowercase letters of the portable character set. When a new file is examined, all marks may
100133 be lost.

100134 Return to Mark

100135 *Synopsis:* '*letter*

100136 Return to the position that was previously marked with the letter named by *letter*, making that
100137 line the current position.

100138 Return to Previous Position

100139 *Synopsis:* ''

100140 Return to the position from which the last large movement command was executed (where a
100141 "large movement" is defined as any movement of more than a screenful of lines). If no such
100142 movements have been made, return to the beginning of the file.

100143 Search Forward for Pattern

100144 *Synopsis:* [*count*]/[!]*pattern*<newline>

100145 Display the screenful beginning with the *count*th line containing the pattern. The search shall
100146 start after the first line currently displayed. The null regular expression ('/' followed by a
100147 <newline>) shall repeat the search using the previous regular expression, with a default *count*. If
100148 the character '!' is included, the matching lines shall be those that do not contain the *pattern*. If
100149 no match is found for the *pattern*, a message to that effect shall be displayed.

100150 Search Backward for Pattern

100151 *Synopsis:* [*count*?][!]*pattern*<newline>

100152 Display the screenful beginning with the *count*th previous line containing the pattern. The search
100153 shall start on the last line before the first line currently displayed. The null regular expression
100154 ('?' followed by a <newline>) shall repeat the search using the previous regular expression,
100155 with a default *count*. If the character '!' is included, matching lines shall be those that do not
100156 contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be
100157 displayed.

100158 Repeat Search

100159 *Synopsis:* [*count*]n

100160 Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last
100161 *pattern*, if the previous search was "/" or "?").

100162 Repeat Search in Reverse

100163 *Synopsis:* [*count*]N

100164 Repeat the search in the opposite direction of the previous search for the *count*th line containing
100165 the last *pattern* (or not containing the last *pattern*, if the previous search was "/" or "?").

100166 **Examine New File**100167 *Synopsis:* :e [*filename*]
<newline>

100168 Examine a new file. If the *filename* argument is not specified, the current file (see the :n and :p
 100169 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell
 100170 word expansions (see Section 2.6, on page 2353); if more than a single pathname results, the
 100171 effects are unspecified. If *filename* is a <number-sign> ('#'), the previously examined file shall
 100172 be re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable
 100173 file), an error message to this effect shall be displayed and the current file and screen shall not
 100174 change.

100175 **Examine Next File**100176 *Synopsis:* [*count*]:n

100177 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If
 100178 *filename* refers to a non-seekable file, the results are unspecified.

100179 **Examine Previous File**100180 *Synopsis:* [*count*]:p

100181 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be
 100182 examined. If *filename* refers to a non-seekable file, the results are unspecified.

100183 **Go to Tag**100184 *Synopsis:* :t *tagstring*
<newline>

100185 If the file containing the tag named by the *tagstring* argument is not the current file, examine the
 100186 file, as if the :e command was executed with that file as the argument. Otherwise, or in addition,
 100187 display the screenful beginning with the tag, as described for the -t option (see the OPTIONS
 100188 section). If the *ctags* utility is not supported by the system, the use of :t produces undefined
 100189 results.

100190 **Invoke Editor**100191 *Synopsis:* v

100192 Invoke an editor to edit the current file being examined. If standard input is being examined, the
 100193 results are unspecified. The name of the editor shall be taken from the environment variable
 100194 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the
 100195 editor shall be invoked with a -c *linenumber* command line argument, where *linenumber* is the
 100196 line number of the file line containing the display line currently displayed as the first line of the
 100197 screen. It is implementation-defined whether line-setting options are passed to editors other
 100198 than *vi* and *ex*.

100199 When the editor exits, *more* shall resume with the same file and screen as when the editor was
 100200 invoked.

100201 **Display Position**
 100202 *Synopsis:* =
 100203 <control>-G

100204 Write a message for which the information references the first byte of the line after the last line of
 100205 the file on the screen. This message shall include the name of the file currently being examined,
 100206 its number relative to the total number of files there are to examine, the line number in the file,
 100207 the byte number and the total bytes in the file, and what percentage of the file precedes the
 100208 current position. If *more* is reading from standard input, or the file is shorter than a single screen,
 100209 the line number, the byte number, the total bytes, and the percentage need not be written.

100210 **Quit**
 100211 *Synopsis:* q
 100212 :q
 100213 ZZ

100214 Exit *more*.

100215 **EXIT STATUS**

100216 The following exit values shall be returned:

100217 0 Successful completion.

100218 >0 An error occurred.

100219 **CONSEQUENCES OF ERRORS**

100220 If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to
 100221 examine the next file in the argument list, but the final exit status shall be affected. If an error is
 100222 encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file
 100223 in the argument list, but the final exit status shall be affected. If an error is encountered accessing
 100224 a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not
 100225 be affected.

100226 **APPLICATION USAGE**

100227 When the standard output is not a terminal, only the **-s** filter-modification option is effective.
 100228 This is based on historical practice. For example, a typical implementation of *man* pipes its
 100229 output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to
 100230 *lp*, however, it is undesirable for this squeezing to happen.

100231 **EXAMPLES**

100232 The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

100233 *more -p G file1 file2*
 100234 Examine each file starting with its last screenful.

100235 *more -p 100 file1 file2*
 100236 Examine each file starting with line 100 in the current position (usually the third line, so line
 100237 98 would be the first line written).

100238 *more -p /100 file1 file2*
 100239 Examine each file starting with the first line containing the string "100" in the current
 100240 position

100241 **RATIONALE**

100242 The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the
 100243 POSIX file display program since it is more widely available than either the public-domain
 100244 program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the
 100245 features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use
 100246 and has become more amenable for *vi* users. Several features originally derived from various file
 100247 editors, found in both *less* and *pg*, have been added to this volume of POSIX.1-2017 as they have
 100248 proved extremely popular with users.

100249 There are inconsistencies between *more* and *vi* that result from historical practice. For example,
 100250 the single-character commands **h**, **f**, **b**, and <space> are screen movers in *more*, but cursor
 100251 movers in *vi*. These inconsistencies were maintained because the cursor movements are not
 100252 applicable to *more* and the powerful functionality achieved without the use of the control key
 100253 justifies the differences.

100254 The tags interface has been included in a program that is not a text editor because it promotes
 100255 another degree of consistent operation with *vi*. It is conceivable that the paging environment of
 100256 *more* would be superior for browsing source code files in some circumstances.

100257 The operating mode referred to for block-mode terminals effectively adds a <newline> to each
 100258 Synopsis line that currently has none. So, for example, **d**<newline> would page one screenful.
 100259 The mode could be triggered by a command line option, environment variable, or some other
 100260 method. The details are not imposed by this volume of POSIX.1-2017 because there are so few
 100261 systems known to support such terminals. Nevertheless, it was considered that all systems
 100262 should be able to support *more* given the exception cited for this small community of terminals
 100263 because, in comparison to *vi*, the cursor movements are few and the command set relatively
 100264 amenable to the optional <newline> characters.

100265 Some versions of *more* provide a shell escaping mechanism similar to the *ex* ! command. The
 100266 standard developers did not consider that this was necessary in a paginator, particularly given
 100267 the wide acceptance of multiple window terminals and job control features. (They chose to
 100268 retain such features in the editors and *mailx* because the shell interaction also gives an
 100269 opportunity to modify the editing buffer, which is not applicable to *more*.)

100270 The **-p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. The
 100271 **+command** option is no longer specified by POSIX.1-2017 but may be present in some
 100272 implementations. In early proposals, it took a *pattern* argument, but historical *less* provided the
 100273 *more* general facility of a command. It would have been desirable to use the same **-c** as *ex* and *vi*,
 100274 but the letter was already in use.

100275 The text stating “from a non-rewindable stream ... implementations may limit the amount of
 100276 backwards motion supported” would allow an implementation that permitted no backwards
 100277 motion beyond text already on the screen. It was not possible to require a minimum amount of
 100278 backwards motion that would be effective for all conceivable device types. The implementation
 100279 should allow the user to back up as far as possible, within device and reasonable memory
 100280 allocation constraints.

100281 Historically, non-printable characters were displayed using the ARPA standard mappings,
 100282 which are as follows:

- 100283 1. Printable characters are left alone.
- 100284 2. Control characters less than \177 are represented as followed by the character offset from
 100285 the '@' character in the ASCII map; for example, \007 is represented as 'G'.

- 100286 3. `\177` is represented as followed by `'?'`.
- 100287 The display of characters having their eighth bit set was less standard. Existing implementations
100288 use hex (`0x00`), octal (`\000`), and a meta-bit display. (The latter displayed characters with their
100289 eighth bit set as the two characters "M-", followed by the seven-bit display as described
100290 previously.) The latter probably has the best claim to historical practice because it was used with
100291 the `-v` option of 4 BSD and 4 BSD-derived versions of the `cat` utility since 1980.
- 100292 No specific display format is required by POSIX.1-2017. Implementations are encouraged to
100293 conform to historic practice in the absence of any strong reason to diverge.
- 100294 **FUTURE DIRECTIONS**
- 100295 None.
- 100296 **SEE ALSO**
- 100297 [Chapter 2](#) (on page 2345), *ctags*, *ed*, *ex*, *vi*
- 100298 [XBD Chapter 8](#) (on page 173), [Section 9.2](#) (on page 182), [Section 9.3](#) (on page 183), [Section 12.2](#)
100299 (on page 216)
- 100300 **CHANGE HISTORY**
- 100301 First released in Issue 4.
- 100302 **Issue 5**
- 100303 The FUTURE DIRECTIONS section is added.
- 100304 **Issue 6**
- 100305 This utility is marked as part of the User Portability Utilities option.
- 100306 The obsolescent SYNOPSIS is removed.
- 100307 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:
- 100308 Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
- 100309 The *more* utility should be able to handle underlined and emboldened displays of
100310 characters that are wider than a single column position.
- 100311 **Issue 7**
- 100312 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized
100313 as an option delimiter in the OPTIONS section.
- 100314 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 100315 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0123 [265] is applied.
- 100316 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0146 [584] is applied.

100317 **NAME**

100318 mv †move files

100319 **SYNOPSIS**100320 mv [-if] *source_file target_file*100321 mv [-if] *source_file... target_dir*100322 **DESCRIPTION**

100323 In the first synopsis form, the *mv* utility shall move the file named by the *source_file* operand to
 100324 the destination specified by the *target_file*. This first synopsis form is assumed when the final
 100325 operand does not name an existing directory and is not a symbolic link referring to an existing
 100326 directory. In this case, if *source_file* names a non-directory file and *target_file* ends with a trailing
 100327 <slash> character, *mv* shall treat this as an error and no *source_file* operands will be processed.

100328 In the second synopsis form, *mv* shall move each file named by a *source_file* operand to a
 100329 destination file in the existing directory named by the *target_dir* operand, or referenced if
 100330 *target_dir* is a symbolic link referring to an existing directory. The destination path for each
 100331 *source_file* shall be the concatenation of the target directory, a single <slash> character if the
 100332 target did not end in a <slash>, and the last pathname component of the *source_file*. This second
 100333 form is assumed when the final operand names an existing directory.

100334 If any operand specifies an existing file of a type not specified by the System Interfaces volume
 100335 of POSIX.1-2017, the behavior is implementation-defined.

100336 For each *source_file* the following steps shall be taken:

100337 1. If the destination path exists, the *-f* option is not specified, and either of the following
 100338 conditions is true:

100339 a. The permissions of the destination path do not permit writing and the standard
 100340 input is a terminal.

100341 b. The *-i* option is specified.

100342 the *mv* utility shall write a prompt to standard error and read a line from standard input.
 100343 If the response is not affirmative, *mv* shall do nothing more with the current *source_file*
 100344 and go on to any remaining *source_files*.

100345 2. If the *source_file* operand and destination path resolve to either the same existing directory
 100346 entry or different directory entries for the same existing file, then the destination path
 100347 shall not be removed, and one of the following shall occur:

100348 a. No change is made to *source_file*, no error occurs, and no diagnostic is issued.

100349 b. No change is made to *source_file*, a diagnostic is issued to standard error
 100350 identifying the two names, and the exit status is affected.

100351 c. If the *source_file* operand and destination path name distinct directory entries, then
 100352 the *source_file* operand is removed, no error occurs, and no diagnostic is issued.

100353 The *mv* utility shall do nothing more with the current *source_file*, and go on to any
 100354 remaining *source_files*.

100355 3. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the
 100356 System Interfaces volume of POSIX.1-2017, called with the following arguments:

100357 a. The *source_file* operand is used as the *old* argument.

- 100358 b. The destination path is used as the *new* argument.
- 100359 If this succeeds, *mv* shall do nothing more with the current *source_file* and go on to any
100360 remaining *source_files*. If this fails for any reasons other than those described for the *errno*
100361 [EXDEV] in the System Interfaces volume of POSIX.1-2017, *mv* shall write a diagnostic
100362 message to standard error, do nothing more with the current *source_file*, and go on to any
100363 remaining *source_files*.
- 100364 4. If the destination path exists, and it is a file of type directory and *source_file* is not a file of
100365 type directory, or it is a file not of type directory and *source_file* is a file of type directory,
100366 *mv* shall write a diagnostic message to standard error, do nothing more with the current
100367 *source_file*, and go on to any remaining *source_files*. If the destination path exists and was
100368 created by a previous step, it is unspecified whether this will be treated as an error or the
100369 destination path will be overwritten.
- 100370 5. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*
100371 shall write a diagnostic message to standard error, do nothing more with the current
100372 *source_file*, and go on to any remaining *source_files*.
- 100373 6. The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the
100374 destination path. If *source_file* or any of the files below it in the hierarchy are symbolic
100375 links, the links themselves shall be duplicated, including their contents, rather than any
100376 files to which they refer. The following characteristics of each file in the file hierarchy
100377 shall be duplicated:
- 100378 The time of last data modification and time of last access
- 100379 The user ID and group ID
- 100380 The file mode
- 100381 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode
100382 bits S_ISUID and S_ISGID shall not be duplicated.
- 100383 When files are duplicated to another file system, the implementation may require that the
100384 process invoking *mv* has read access to each file being duplicated.
- 100385 If files being duplicated to another file system have hard links to other files, it is
100386 unspecified whether the files copied to the new file system have the hard links preserved or
100387 separate copies are created for the linked files.
- 100388 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic
100389 message to standard error, do nothing more with the current *source_file*, and go on to any
100390 remaining *source_files*.
- 100391 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic
100392 message to standard error, but this failure shall not cause *mv* to modify its exit status.
- 100393 7. The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, *mv*
100394 shall write a diagnostic message to the standard error, do nothing more with the current
100395 *source_file*, and go on to any remaining *source_files*.

100396 OPTIONS

- 100397 The *mv* utility shall conform to XBD [Section 12.2](#) (on page 216).
- 100398 The following options shall be supported:
- 100399 **-f** Do not prompt for confirmation if the destination path exists. Any previous
100400 occurrence of the **-i** option is ignored.

- 100401 **-i** Prompt for confirmation if the destination path exists. Any previous occurrence of
100402 the **-f** option is ignored.
- 100403 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option
100404 specified shall determine the behavior of *mv*.
- 100405 **OPERANDS**
- 100406 The following operands shall be supported:
- 100407 *source_file* A pathname of a file or directory to be moved.
- 100408 *target_file* A new pathname for the file or directory being moved.
- 100409 *target_dir* A pathname of an existing directory into which to move the input files.
- 100410 **STDIN**
- 100411 The standard input shall be used to read an input line in response to each prompt specified in
100412 the **STDERR** section. Otherwise, the standard input shall not be used.
- 100413 **INPUT FILES**
- 100414 The input files specified by each *source_file* operand can be of any file type.
- 100415 **ENVIRONMENT VARIABLES**
- 100416 The following environment variables shall affect the execution of *mv*:
- 100417 **LANG** Provide a default value for the internationalization variables that are unset or null.
100418 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
100419 variables used to determine the values of locale categories.)
- 100420 **LC_ALL** If set to a non-empty string value, override the values of all the other
100421 internationalization variables.
- 100422 **LC_COLLATE**
- 100423 Determine the locale for the behavior of ranges, equivalence classes, and multi-
100424 character collating elements used in the extended regular expression defined for
100425 the **yesexpr** locale keyword in the **LC_MESSAGES** category.
- 100426 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
100427 characters (for example, single-byte as opposed to multi-byte characters in
100428 arguments and input files), the behavior of character classes used in the extended
100429 regular expression defined for the **yesexpr** locale keyword in the **LC_MESSAGES**
100430 category.
- 100431 **LC_MESSAGES**
- 100432 Determine the locale used to process affirmative responses, and the locale used to
100433 affect the format and contents of diagnostic messages and prompts written to
100434 standard error.
- 100435 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 100436 **ASYNCHRONOUS EVENTS**
- 100437 Default.
- 100438 **STDOUT**
- 100439 Not used.
- 100440 **STDERR**
- 100441 Prompts shall be written to the standard error under the conditions specified in the
100442 **DESCRIPTION** section. The prompts shall contain the destination pathname, but their format is
100443 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic

100444 messages.

100445 OUTPUT FILES

100446 The output files may be of any file type.

100447 EXTENDED DESCRIPTION

100448 None.

100449 EXIT STATUS

100450 The following exit values shall be returned:

100451 0 All input files were moved successfully.

100452 >0 An error occurred.

100453 CONSEQUENCES OF ERRORS

100454 If the copying or removal of *source_file* is prematurely terminated by a signal or error, *mv* may
 100455 leave a partial copy of *source_file* at the source or destination. The *mv* utility shall not modify
 100456 both *source_file* and the destination path simultaneously; termination at any point shall leave
 100457 either *source_file* or the destination path complete.

100458 APPLICATION USAGE

100459 Some implementations mark for update the last file status change timestamp of renamed files
 100460 and some do not. Applications which make use of the last file status change timestamp may
 100461 behave differently with respect to renamed files unless they are designed to allow for either
 100462 behavior.

100463 The specification ensures that *mv a a* will not alter the contents of file **a**, and allows the
 100464 implementation to issue an error that a file cannot be moved onto itself. Likewise, when **a** and **b**
 100465 are hard links to the same file, *mv a b* will not alter **b**, but if a diagnostic is not issued, then it is
 100466 unspecified whether **a** is left untouched (as it would be by the *rename()* function) or unlinked
 100467 (reducing the link count of **b**).

100468 EXAMPLES

100469 If the current directory contains only files **a** (of any type defined by the System Interfaces
 100470 volume of POSIX.1-2017), **b** (also of any type), and a directory **c**:

100471 `mv a b c`

100472 `mv c d`

100473 results with the original files **a** and **b** residing in the directory **d** in the current directory.

100474 RATIONALE

100475 Early proposals diverged from the SVID and BSD historical practice in that they required that
 100476 when the destination path exists, the `-f` option is not specified, and input is not a terminal, *mv*
 100477 fails. This was done for compatibility with *cp*. The current text returns to historical practice. It
 100478 should be noted that this is consistent with the *rename()* function defined in the System
 100479 Interfaces volume of POSIX.1-2017, which does not require write permission on the target.

100480 For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for
 100481 confirmation, should be interpreted in the following manner:

```
100482 if (exists AND (NOT f_option) AND
100483     ((not_writable AND input_is_terminal) OR i_option))
```

100484 The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally
 100485 unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv*
 100486 deletes all existing destination paths without prompting, even when `-i` is specified; this is
 100487 inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when

100488 the file is unwritable and the standard input is not a terminal. The standard developers decided
100489 that use of `-i` is a request for interaction, so when the destination path exists, the utility takes
100490 instructions from whatever responds to standard input.

100491 The `rename()` function is able to move directories within the same file system. Some historical
100492 versions of `mv` have been able to move directories, but not to a different file system. The
100493 standard developers considered that this was an annoying inconsistency, so this volume of
100494 POSIX.1-2017 requires directories to be able to be moved even across file systems. There is no `-R`
100495 option to confirm that moving a directory is actually intended, since such an option was not
100496 required for moving directories in historical practice. Requiring the application to specify it
100497 sometimes, depending on the destination, seemed just as inconsistent. The semantics of the
100498 `rename()` function were preserved as much as possible. For example, `mv` is not permitted to
100499 “rename” files to or from directories, even though they might be empty and removable.

100500 Historic implementations of `mv` did not exit with a non-zero exit status if they were unable to
100501 duplicate any file characteristics when moving a file across file systems, nor did they write a
100502 diagnostic message for the user. The former behavior has been preserved to prevent scripts from
100503 breaking; a diagnostic message is now required, however, so that users are alerted that the file
100504 characteristics have changed.

100505 The exact format of the interactive prompts is unspecified. Only the general nature of the
100506 contents of prompts are specified because implementations may desire more descriptive
100507 prompts than those used on historical implementations. Therefore, an application not using the
100508 `-f` option or using the `-i` option relies on the system to provide the most suitable dialog directly
100509 with the user, based on the behavior specified.

100510 When `mv` is dealing with a single file system and `source_file` is a symbolic link, the link itself is
100511 moved as a consequence of the dependence on the `rename()` functionality, per the
100512 DESCRIPTION. Across file systems, this has to be made explicit.

100513 **FUTURE DIRECTIONS**

100514 None.

100515 **SEE ALSO**

100516 [*cp*](#), [*ln*](#)

100517 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

100518 XSH [*rename\(\)*](#)

100519 **CHANGE HISTORY**

100520 First released in Issue 2.

100521 **Issue 6**

100522 The `mv` utility is changed to describe processing of symbolic links as specified in the
100523 IEEE P1003.2b draft standard.

100524 The APPLICATION USAGE section is added.

100525 **Issue 7**

100526 Austin Group Interpretation 1003.1-2001 #016 is applied.

100527 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
100528 `LC_MESSAGES` environment variable.

100529 Austin Group Interpretations 1003.1-2001 #164, #168, and #169 are applied.

100530 SD5-XCU-ERN-13 is applied, making an editorial correction to the SYNOPSIS.

100531 SD5-XCU-ERN-51 is applied to the DESCRIPTION, defining the behavior for when files are

- 100532 being duplicated to another file system while having hard links.
- 100533 Changes are made related to support for finegrained timestamps.
- 100534 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0124 [48] is applied.
- 100535 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0147 [534] is applied.

100536 **NAME**

100537 newgrp — change to a new group

100538 **SYNOPSIS**100539 newgrp [-l] [*group*]100540 **DESCRIPTION**

100541 The *newgrp* utility shall create a new shell execution environment with a new real and effective
 100542 group identification. Of the attributes listed in [Section 2.12](#) (on page 2381), the new shell
 100543 execution environment shall retain the working directory, file creation mask, and exported
 100544 variables from the previous environment (that is, open files, traps, unexported variables, alias
 100545 definitions, shell functions, and *set* options may be lost). All other aspects of the process
 100546 environment that are preserved by the *exec* family of functions defined in the System Interfaces
 100547 volume of POSIX.1-2017 shall also be preserved by *newgrp*; whether other aspects are preserved
 100548 is unspecified.

100549 A failure to assign the new group identifications (for example, for security or password-related
 100550 reasons) shall not prevent the new shell execution environment from being created.

100551 The *newgrp* utility shall affect the supplemental groups for the process as follows:

100552 On systems where the effective group ID is normally in the supplementary group list (or
 100553 whenever the old effective group ID actually is in the supplementary group list):

100554 If the new effective group ID is also in the supplementary group list, *newgrp* shall
 100555 change the effective group ID.

100556 If the new effective group ID is not in the supplementary group list, *newgrp* shall add
 100557 the new effective group ID to the list, if there is room to add it.

100558 On systems where the effective group ID is not normally in the supplementary group list
 100559 (or whenever the old effective group ID is not in the supplementary group list):

100560 If the new effective group ID is in the supplementary group list, *newgrp* shall delete
 100561 it.

100562 If the old effective group ID is not in the supplementary list, *newgrp* shall add it if
 100563 there is room.

100564 **Note:** The System Interfaces volume of POSIX.1-2017 does not specify whether the effective group ID
 100565 of a process is included in its supplementary group list.

100566 With no operands, *newgrp* shall change the effective group back to the groups identified in the
 100567 user's user entry, and shall set the list of supplementary groups to that set in the user's group
 100568 database entries.

100569 If the first argument is '-', the results are unspecified.

100570 If a password is required for the specified group, and the user is not listed as a member of that
 100571 group in the group database, the user shall be prompted to enter the correct password for that
 100572 group. If the user is listed as a member of that group, no password shall be requested. If no
 100573 password is required for the specified group, it is implementation-defined whether users not
 100574 listed as members of that group can change to that group. Whether or not a password is
 100575 required, implementation-defined system accounting or security mechanisms may impose
 100576 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and
 100577 suppress the changing of the group identification.

100578 **OPTIONS**

100579 The *newgrp* utility shall conform to XBD [Section 12.2](#) (on page 216), except for the unspecified
100580 usage of '-'.

100581 The following option shall be supported:

100582 **-l** (The letter ell.) Change the environment to what would be expected if the user
100583 actually logged in again.

100584 **OPERANDS**

100585 The following operand shall be supported:

100586 *group* A group name from the group database or a non-negative numeric group ID.
100587 Specifies the group ID to which the real and effective group IDs shall be set. If
100588 *group* is a non-negative numeric string and exists in the group database as a group
100589 name (see *getgrnam()*), the numeric group ID associated with that group name
100590 shall be used as the group ID.

100591 **STDIN**

100592 Not used.

100593 **INPUT FILES**

100594 The file */dev/tty* shall be used to read a single line of text for password checking, when one is
100595 required.

100596 **ENVIRONMENT VARIABLES**

100597 The following environment variables shall affect the execution of *newgrp*:

100598 *LANG* Provide a default value for the internationalization variables that are unset or null.
100599 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
100600 variables used to determine the values of locale categories.)

100601 *LC_ALL* If set to a non-empty string value, override the values of all the other
100602 internationalization variables.

100603 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
100604 characters (for example, single-byte as opposed to multi-byte characters in
100605 arguments).

100606 *LC_MESSAGES*

100607 Determine the locale that should be used to affect the format and contents of
100608 diagnostic messages written to standard error.

100609 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

100610 **ASYNCHRONOUS EVENTS**

100611 Default.

100612 **STDOUT**

100613 Not used.

100614 **STDERR**

100615 The standard error shall be used for diagnostic messages and a prompt string for a password, if
100616 one is required. Diagnostic messages may be written in cases where the exit status is not
100617 available. See the EXIT STATUS section.

100618 **OUTPUT FILES**

100619 None.

100620 **EXTENDED DESCRIPTION**

100621 None.

100622 **EXIT STATUS**

100623 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group
100624 identification was changed successfully, the exit status shall be the exit status of the shell.
100625 Otherwise, the following exit value shall be returned:

100626 >0 An error occurred.

100627 **CONSEQUENCES OF ERRORS**

100628 The invoking shell may terminate.

100629 **APPLICATION USAGE**

100630 There is no convenient way to enter a password into the group database. Use of group
100631 passwords is not encouraged, because by their very nature they encourage poor security
100632 practices. Group passwords may disappear in the future.

100633 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with
100634 *newgrp*, which in turn overlays itself with a new shell after changing group. On some
100635 implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

100636 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a
100637 useful interface for the support of applications.

100638 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it
100639 successfully invokes a new shell and the rest of the original shell script is bypassed when the
100640 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.
100641 But usage such as:

100642 `newgrp foo`100643 `echo $?`

100644 is not useful because the new shell might not have access to any status *newgrp* may have
100645 generated (and most historical systems do not provide this status). A zero status echoed here
100646 does not necessarily indicate that the user has changed to the new group successfully. Following
100647 *newgrp* with the *id* command provides a portable means of determining whether the group
100648 change was successful or not.

100649 **EXAMPLES**

100650 None.

100651 **RATIONALE**

100652 Most historical implementations use one of the *exec* functions to implement the behavior of
100653 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected
100654 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*
100655 issue a diagnostic message to tell the user that the environment changed, it would be
100656 inappropriate to require this change to some historical implementations.

100657 The password mechanism is allowed in the group database, but how this would be
100658 implemented is not specified.

100659 The *newgrp* utility was retained in this volume of POSIX.1-2017, even given the existence of the
100660 multiple group permissions feature in the System Interfaces volume of POSIX.1-2017, for several
100661 reasons. First, in some implementations, the group ownership of a newly created file is
100662 determined by the group of the directory in which the file is created, as allowed by the System

100663 Interfaces volume of POSIX.1-2017; on other implementations, the group ownership of a newly
100664 created file is determined by the effective group ID. On implementations of the latter type,
100665 *newgrp* allows files to be created with a specific group ownership. Finally, many
100666 implementations use the real group ID in accounting, and on such systems, *newgrp* allows the
100667 accounting identity of the user to be changed.

100668 **FUTURE DIRECTIONS**

100669 None.

100670 **SEE ALSO**

100671 [Chapter 2](#) (on page 2345), *sh*

100672 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

100673 XSH *exec*, *getgrnam()*

100674 **CHANGE HISTORY**

100675 First released in Issue 2.

100676 **Issue 6**

100677 This utility is marked as part of the User Portability Utilities option.

100678 The obsolescent SYNOPSIS is removed.

100679 The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being
100680 greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS
100681 requirement.

100682 **Issue 7**

100683 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
100684 argument is '- '.

100685 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100686 The *newgrp* utility is moved from the User Portability Utilities option to the Base. User
100687 Portability Utilities is now an option for interactive utilities.

100688 **NAME**

100689 nice — invoke a utility with an altered nice value

100690 **SYNOPSIS**

100691 nice [-n *increment*] *utility* [*argument...*]

100692 **DESCRIPTION**

100693 The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see
 100694 XBD [Section 3.244](#), on page 72). With no options, the executed utility shall be run with a nice
 100695 value that is some implementation-defined quantity greater than or equal to the nice value of the
 100696 current process. If the user lacks appropriate privileges to affect the nice value in the requested
 100697 manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be
 100698 written to standard error, but this shall not prevent the invocation of *utility* or affect the exit
 100699 status.

100700 **OPTIONS**

100701 The *nice* utility shall conform to XBD [Section 12.2](#) (on page 216).

100702 The following option is supported:

100703 **-n *increment*** A positive or negative decimal integer which shall have the same effect on the
 100704 execution of the utility as if the utility had called the *nice()* function with the
 100705 numeric value of the *increment* option-argument.

100706 **OPERANDS**

100707 The following operands shall be supported:

100708 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 100709 special built-in utilities in [Section 2.14](#) (on page 2384), the results are undefined.

100710 *argument* Any string to be supplied as an argument when invoking the utility named by the
 100711 *utility* operand.

100712 **STDIN**

100713 Not used.

100714 **INPUT FILES**

100715 None.

100716 **ENVIRONMENT VARIABLES**

100717 The following environment variables shall affect the execution of *nice*:

100718 **LANG** Provide a default value for the internationalization variables that are unset or null.
 100719 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 100720 variables used to determine the values of locale categories.)

100721 **LC_ALL** If set to a non-empty string value, override the values of all the other
 100722 internationalization variables.

100723 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 100724 characters (for example, single-byte as opposed to multi-byte characters in
 100725 arguments).

100726 **LC_MESSAGES**

100727 Determine the locale that should be used to affect the format and contents of
 100728 diagnostic messages written to standard error.

100729 **XSIX** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

100730 *PATH* Determine the search path used to locate the utility to be invoked. See XBD
100731 [Chapter 8](#) (on page 173).

100732 **ASYNCHRONOUS EVENTS**

100733 Default.

100734 **STDOUT**

100735 Not used.

100736 **STDERR**

100737 The standard error shall be used only for diagnostic messages.

100738 **OUTPUT FILES**

100739 None.

100740 **EXTENDED DESCRIPTION**

100741 None.

100742 **EXIT STATUS**

100743 If *utility* is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise, the *nice*
100744 utility shall exit with one of the following values:

100745 1-125 An error occurred in the *nice* utility.

100746 126 The utility specified by *utility* was found but could not be invoked.

100747 127 The utility specified by *utility* could not be found.

100748 **CONSEQUENCES OF ERRORS**

100749 Default.

100750 **APPLICATION USAGE**

100751 The only guaranteed portable uses of this utility are:

100752 *nice utility*

100753 Run *utility* with the default higher or equal nice value.

100754 *nice -n <positive integer> utility*

100755 Run *utility* with a higher nice value.

100756 On some implementations they have no discernible effect on the invoked utility and on some
100757 others they are exactly equivalent.

100758 Historical systems have frequently supported the *<positive integer>* up to 20. Since there is no
100759 error penalty associated with guessing a number that is too high, users without access to the
100760 system conformance document (to see what limits are actually in place) could use the historical 1
100761 to 20 range or attempt to use very large numbers if the job should be truly low priority.

100762 The nice value of a process can be displayed using the command:

100763 `ps -o nice`

100764 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
100765 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
100766 utility exited with an error indication”. The value 127 was chosen because it is not commonly
100767 used for other meanings; most utilities use small values for “normal error conditions” and the
100768 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
100769 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
100770 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
100771 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
100772 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for

100773 any other reason.

100774 **EXAMPLES**

100775 None.

100776 **RATIONALE**

100777 The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The
100778 command *nice -x utility*, for example, would be treated the same as the command *nice --1*
100779 *utility*. If the user does not have appropriate privileges, this results in a “permission denied”
100780 error. This is considered a bug.

100781 When a user without appropriate privileges gives a negative *increment*, System V treats it like
100782 the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not
100783 run the utility. The standard specifies the System V behavior together with an optional BSD-style
100784 “permission denied” message.

100785 The C shell has a built-in version of *nice* that has a different interface from the one described in
100786 this volume of POSIX.1-2017.

100787 The term “utility” is used, rather than “command”, to highlight the fact that shell compound
100788 commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used.
100789 However, “utility” includes user application programs and shell scripts, not just utilities defined
100790 in this volume of POSIX.1-2017.

100791 Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the
100792 default nice value being the midpoint of that range. By default, they raise the nice value of the
100793 executed utility by 10.

100794 Some historical documentation states that the *increment* value must be within a fixed range. This
100795 is misleading; the valid *increment* values on any invocation are determined by the current
100796 process nice value, which is not always the default.

100797 The definition of nice value is not intended to suggest that all processes in a system have
100798 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the
100799 System Interfaces volume of POSIX.1-2017 make the notion of a single underlying priority for all
100800 scheduling policies problematic. Some implementations may implement the *nice*-related features
100801 to affect all processes on the system, others to affect just the general time-sharing activities
100802 implied by this volume of POSIX.1-2017, and others may have no effect at all. Because of the use
100803 of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are
100804 possible.

100805 Earlier versions of this standard allowed a *-increment* option. This form is no longer specified by
100806 POSIX.1-2017 but may be present in some implementations.

100807 **FUTURE DIRECTIONS**

100808 None.

100809 **SEE ALSO**

100810 [Chapter 2](#) (on page 2345), [renice](#)

100811 [XBD Section 3.244](#) (on page 72), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

100812 XSH [nice\(\)](#)

100813 **CHANGE HISTORY**

100814 First released in Issue 4.

100815 **Issue 6**

100816 This utility is marked as part of the User Portability Utilities option.

100817 The obsolescent SYNOPSIS is removed.

100818 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/18 is applied, deleting a paragraph of
100819 RATIONALE that referred to text no longer in the standard.

100820 **Issue 7**

100821 Austin Group Interpretation 1003.1-2001 #027 is applied.

100822 SD5-XCU-ERN-32 and SD5-XCU-ERN-33 are applied, updating the DESCRIPTION,
100823 APPLICATION USAGE, and RATIONALE sections.

100824 The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability
100825 Utilities is now an option for interactive utilities.

100826 **NAME**100827 nl *†*line numbering filter100828 **SYNOPSIS**

```
100829 XSI nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num]
100830 [-n format] [-s sep] [-v startnum] [-w width] [file]
```

100831 **DESCRIPTION**

100832 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and
 100833 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional
 100834 functionality may be provided in accordance with the command options in effect.

100835 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at
 100836 the start of each logical page. A logical page consists of a header, a body, and a footer section.
 100837 Empty sections are valid. Different line numbering options are independently available for
 100838 header, body, and footer (for example, no numbering of header and footer lines while
 100839 numbering blank lines only in the body).

100840 The starts of logical page sections shall be signaled by input lines containing nothing but the
 100841 following delimiter characters:

Line	Start of
\: \: \:	Header
\: \:	Body
\:	Footer

100842 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

100847 **OPTIONS**

100848 The *nl* utility shall conform to XBD [Section 12.2](#) (on page 216). Only one file can be named.

100849 The following options shall be supported:

100850 **-b *type*** Specify which logical page body lines shall be numbered. Recognized *types* and
 100851 their meaning are:

100852 **a** Number all lines.

100853 **t** Number only non-empty lines.

100854 **n** No line numbering.

100855 **pstring** Number only lines that contain the basic regular expression specified in
 100856 *string*.

100857 The default *type* for logical page body shall be **t** (text lines numbered).

100858 **-d *delim*** Specify the delimiter characters that indicate the start of a logical page section.
 100859 These can be changed from the default characters "\:" to two user-specified
 100860 characters. If only one character is entered, the second character shall remain the
 100861 default character ' : '.

100862 **-f *type*** Specify the same as **b *type*** except for footer. The default for logical page footer shall
 100863 be **n** (no lines numbered).

100864 **-h *type*** Specify the same as **b *type*** except for header. The default *type* for logical page
 100865 header shall be **n** (no lines numbered).

100866 **-i** *incr* Specify the increment value used to number logical page lines. The default shall be
100867 1.

100868 **-l** *num* Specify the number of blank lines to be considered as one. For example, **-l 2** results
100869 in only the second adjacent blank line being numbered (if the appropriate **-h a**,
100870 **-b a**, or **-f a** option is set). The default shall be 1.

100871 **-n** *format* Specify the line numbering format. Recognized values are: **ln**, left justified, leading
100872 zeros suppressed; **rn**, right justified, leading zeros suppressed; **rz**, right justified,
100873 leading zeros kept. The default *format* shall be **rn** (right justified).

100874 **-p** Specify that numbering should not be restarted at logical page delimiters.

100875 **-s** *sep* Specify the characters used in separating the line number and the corresponding
100876 text line. The default *sep* shall be a <tab>.

100877 **-v** *startnum* Specify the initial value used to number logical page lines. The default shall be 1.

100878 **-w** *width* Specify the number of characters to be used for the line number. The default *width*
100879 shall be 6.

100880 OPERANDS

100881 The following operand shall be supported:

100882 *file* A pathname of a text file to be line-numbered.

100883 STDIN

100884 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*
100885 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
100886 the standard input shall not be used. See the INPUT FILES section.

100887 INPUT FILES

100888 The input file shall be a text file.

100889 ENVIRONMENT VARIABLES

100890 The following environment variables shall affect the execution of *nl*:

100891 **LANG** Provide a default value for the internationalization variables that are unset or null.
100892 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
100893 variables used to determine the values of locale categories.)

100894 **LC_ALL** If set to a non-empty string value, override the values of all the other
100895 internationalization variables.

100896 **LC_COLLATE**

100897 Determine the locale for the behavior of ranges, equivalence classes, and multi-
100898 character collating elements within regular expressions.

100899 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
100900 characters (for example, single-byte as opposed to multi-byte characters in
100901 arguments and input files), the behavior of character classes within regular
100902 expressions, and for deciding which characters are in character class **graph** (for the
100903 **-b t**, **-f t**, and **-h t** options).

100904 **LC_MESSAGES**

100905 Determine the locale that should be used to affect the format and contents of
100906 diagnostic messages written to standard error.

- 100907 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 100908 **ASYNCHRONOUS EVENTS**
- 100909 Default.
- 100910 **STDOUT**
- 100911 The standard output shall be a text file in the following format:
- 100912 "%s%s%s", *<line number>*, *<separator>*, *<input line>*
- 100913 where *<line number>* is one of the following numeric formats:
- 100914 %6d When the **rn** format is used (the default; see **-n**).
- 100915 %06d When the **rz** format is used.
- 100916 %-6d When the **ln** format is used.
- 100917 *<empty>* When line numbers are suppressed for a portion of the page; the *<separator>* is also
100918 suppressed.
- 100919 In the preceding list, the number 6 is the default width; the **-w** option can change this value.
- 100920 **STDERR**
- 100921 The standard error shall be used only for diagnostic messages.
- 100922 **OUTPUT FILES**
- 100923 None.
- 100924 **EXTENDED DESCRIPTION**
- 100925 None.
- 100926 **EXIT STATUS**
- 100927 The following exit values shall be returned:
- 100928 0 Successful completion.
- 100929 >0 An error occurred.
- 100930 **CONSEQUENCES OF ERRORS**
- 100931 Default.
- 100932 **APPLICATION USAGE**
- 100933 In using the **-d** *delim* option, care should be taken to escape characters that have special meaning
100934 to the command interpreter.
- 100935 **EXAMPLES**
- 100936 The command:
- 100937 `nl -v 10 -i 10 -d \!+ file1`
- 100938 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is
100939 "!+". Note that the "!" has to be escaped when using *cs*h as a command interpreter because of
100940 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any
100941 harm.
- 100942 **RATIONALE**
- 100943 None.

100944 **FUTURE DIRECTIONS**

100945 None.

100946 **SEE ALSO**100947 *pr*100948 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)100949 **CHANGE HISTORY**

100950 First released in Issue 2.

100951 **Issue 5**100952 The option [-f *type*] is added to the SYNOPSIS. The option descriptions are presented in
100953 alphabetic order. The description of -bt is changed to ``Number only non-empty lines''.100954 **Issue 6**100955 The obsolescent behavior allowing the options to be intermingled with the optional *file* operand
100956 is removed.100957 **Issue 7**

100958 Austin Group Interpretation 1003.1-2001 #092 is applied.

100959 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100960 **NAME**

100961 nm `[-w]` write the name list of an object file (DEVELOPMENT)

100962 **SYNOPSIS**

100963 SD nm `[-APv] [-g|-u] [-t format] file...`

100964 XSI nm `[-APv] [-efox] [-g|-u] [-t format] file...`

100965 **DESCRIPTION**

100966 The *nm* utility shall display symbolic information appearing in the object file, executable file, or
 100967 object-file library named by *file*. If no symbolic information is available for a valid input file, the
 100968 *nm* utility shall report that fact, but not consider it an error condition.

100969 XSI The default base used when numeric values are written is unspecified. On XSI-conformant
 100970 systems, it shall be decimal if the `-P` option is not specified.

100971 **OPTIONS**

100972 The *nm* utility shall conform to XBD Section 12.2 (on page 216).

100973 The following options shall be supported:

100974 `-A` Write the full pathname or library name of an object on each line.

100975 XSI `-e` Write only external (global) and static symbol information.

100976 XSI `-f` Produce full output. Write redundant symbols (`.text`, `.data`, and `.bss`), normally
 100977 suppressed.

100978 `-g` Write only external (global) symbol information.

100979 XSI `-o` Write numeric values in octal (equivalent to `-t o`).

100980 `-P` Write information in a portable output format, as specified in the STDOUT section.

100981 `-t format` Write each numeric value in the specified format. The format shall be dependent
 100982 on the single character used as the *format* option-argument:

100983 XSI `d` decimal (default if `-P` is not specified).

100984 `o` octal.

100985 `x` hexadecimal (default if `-P` is specified).

100986 `-u` Write only undefined symbols.

100987 `-v` Sort output by value instead of by symbol name.

100988 XSI `-x` Write numeric values in hexadecimal (equivalent to `-t x`).

100989 **OPERANDS**

100990 The following operand shall be supported:

100991 *file* A pathname of an object file, executable file, or object-file library.

100992 **STDIN**

100993 See the INPUT FILES section.

100994 **INPUT FILES**

100995 The input file shall be an object file, an object-file library whose format is the same as those
 100996 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept
 100997 additional implementation-defined object library formats for the input file.

100998 **ENVIRONMENT VARIABLES**

100999 The following environment variables shall affect the execution of *nm*:

101000 *LANG* Provide a default value for the internationalization variables that are unset or null.
 101001 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 101002 variables used to determine the values of locale categories.)

101003 *LC_ALL* If set to a non-empty string value, override the values of all the other
 101004 internationalization variables.

101005 *LC_COLLATE*

101006 Determine the locale for character collation information for the symbol-name and
 101007 symbol-value collation sequences.

101008 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101009 characters (for example, single-byte as opposed to multi-byte characters in
 101010 arguments).

101011 *LC_MESSAGES*

101012 Determine the locale that should be used to affect the format and contents of
 101013 diagnostic messages written to standard error.

101014 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

101015 **ASYNCHRONOUS EVENTS**

101016 Default.

101017 **STDOUT**

101018 If symbolic information is present in the input files, then for each file or for each member of an
 101019 archive, the *nm* utility shall write the following information to standard output. By default, the
 101020 format is unspecified, but the output shall be sorted by symbol name according to the collation
 101021 sequence in the current locale.

101022 Library or object name, if *-A* is specified

101023 Symbol name

101024 Symbol type, which shall either be one of the following single characters or an
 101025 implementation-defined type represented by a single character:

101026 A Global absolute symbol.

101027 a Local absolute symbol.

101028 B Global ``bss'' (that is, uninitialized data space) symbol.

101029 b Local bss symbol.

101030 D Global data symbol.

101031 d Local data symbol.

101032 T Global text symbol.

101033 t Local text symbol.

101034 U Undefined symbol.

101035 Value of the symbol

101036 The size associated with the symbol, if applicable

101037 This information may be supplemented by additional information specific to the

101038 implementation.

101039 If the **-P** option is specified, the previous information shall be displayed using the following
 101040 portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,
 101041 respectively:

101042 "%s%s %s %d %d\n", <library/object name>, <name>, <type>,
 101043 <value>, <size>

101044 "%s%s %s %o %o\n", <library/object name>, <name>, <type>,
 101045 <value>, <size>

101046 "%s%s %s %x %x\n", <library/object name>, <name>, <type>,
 101047 <value>, <size>

101048 where <library/object name> shall be formatted as follows:

101049 If **-A** is not specified, <library/object name> shall be an empty string.

101050 If **-A** is specified and the corresponding *file* operand does not name a library:

101051 "%s: ", <file>

101052 If **-A** is specified and the corresponding *file* operand names a library. In this case,
 101053 <object file> shall name the object file in the library containing the symbol being described:

101054 "%s[%s]: ", <file>, <object file>

101055 If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is
 101056 specified and it names a library, *nm* shall write a line identifying the object containing the
 101057 following symbols before the lines containing those symbols, in the form:

101058 If the corresponding *file* operand does not name a library:

101059 "%s:\n", <file>

101060 If the corresponding *file* operand names a library; in this case, <object file> shall be the
 101061 name of the file in the library containing the following symbols:

101062 "%s[%s]:\n", <file>, <object file>

101063 If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

101064 **STDERR**

101065 The standard error shall be used only for diagnostic messages.

101066 **OUTPUT FILES**

101067 None.

101068 **EXTENDED DESCRIPTION**

101069 None.

101070 **EXIT STATUS**

101071 The following exit values shall be returned:

101072 0 Successful completion.

101073 >0 An error occurred.

101074 **CONSEQUENCES OF ERRORS**

101075 Default.

101076 APPLICATION USAGE

101077 Mechanisms for dynamic linking make this utility less meaningful when applied to an
101078 executable file because a dynamically linked executable may omit numerous library routines
101079 that would be found in a statically linked executable.

101080 EXAMPLES

101081 None.

101082 RATIONALE

101083 Historical implementations of *nm* have used different bases for numeric output and supplied
101084 different default types of symbols that were reported. The `-t format` option, similar to that used
101085 in *od* and *strings*, can be used to specify the numeric base; `-g` and `-u` can be used to restrict the
101086 amount of output or the types of symbols included in the output.

101087 The compromise of using `-t format` versus using `-d`, `-o`, and other similar options was necessary
101088 because of differences in the meaning of `-o` between implementations. The `-o` option from BSD
101089 has been provided here as `-A` to avoid confusion with the `-o` from System V (which has been
101090 provided here as `-t` and as `-o` on XSI-conformant systems).

101091 The option list was significantly reduced from that provided by historical implementations.

101092 The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified
101093 default output.

101094 It was recognized that mechanisms for dynamic linking make this utility less meaningful when
101095 applied to an executable file (because a dynamically linked executable file may omit numerous
101096 library routines that would be found in a statically linked executable file), but the value of *nm*
101097 during software development was judged to outweigh other limitations.

101098 The default output format of *nm* is not specified because of differences in historical
101099 implementations. The `-P` option was added to allow some type of portable output format. After
101100 a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to
101101 create one that did not match the current format of any of these four systems. The format
101102 devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary
101103 depending on locale (because no English descriptions are included). All of the systems currently
101104 have the information available to use this format.

101105 The format given in *nm* STDOUT uses `<space>` characters between the fields, which may be any
101106 number of `<blank>` characters required to align the columns. The single-character types were
101107 selected to match historical practice, and the requirement that implementation additions also be
101108 single characters made parsing the information easier for shell scripts.

101109 FUTURE DIRECTIONS

101110 None.

101111 SEE ALSO

101112 [ar](#), [c99](#)

101113 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

101114 CHANGE HISTORY

101115 First released in Issue 2.

101116 Issue 6

101117 This utility is marked as supported when both the User Portability Utilities option and the
101118 Software Development Utilities option are supported.

101119 **Issue 7**

- 101120 The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.
- 101121
- 101122 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 101123 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0125 [263] and XCU/TC1-2008/0126 [263] are applied.
- 101124
- 101125 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0148 [744] is applied.

101126 **NAME**101127 nohup \ddagger invoke a utility immune to hangups101128 **SYNOPSIS**101129 nohup *utility* [*argument...*]101130 **DESCRIPTION**

101131 The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied
 101132 as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be
 101133 set to be ignored.

101134 If standard input is associated with a terminal, the *nohup* utility may redirect standard input
 101135 from an unspecified file.

101136 If the standard output is a terminal, all output written by the named *utility* to its standard output
 101137 shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot
 101138 be created or opened for appending, the output shall be appended to the end of the file
 101139 **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be
 101140 created or opened for appending, *utility* shall not be invoked. If a file is created, the file's
 101141 permission bits shall be set to S_IRUSR | S_IWUSR.

101142 If standard error is a terminal and standard output is open but is not a terminal, all output
 101143 written by the named utility to its standard error shall be redirected to the same open file
 101144 description as the standard output. If standard error is a terminal and standard output either is a
 101145 terminal or is closed, the same output shall instead be appended to the end of the **nohup.out** file
 101146 as described above.

101147 **OPTIONS**

101148 None.

101149 **OPERANDS**

101150 The following operands shall be supported:

101151 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 101152 special built-in utilities in [Section 2.14](#) (on page 2384), the results are undefined.

101153 *argument* Any string to be supplied as an argument when invoking the utility named by the
 101154 *utility* operand.

101155 **STDIN**

101156 Not used.

101157 **INPUT FILES**

101158 None.

101159 **ENVIRONMENT VARIABLES**101160 The following environment variables shall affect the execution of *nohup*:

101161 *HOME* Determine the pathname of the user's home directory: if the output file **nohup.out**
 101162 cannot be created in the current directory, the *nohup* utility shall use the directory
 101163 named by *HOME* to create the file.

101164 *LANG* Provide a default value for the internationalization variables that are unset or null.
 101165 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 101166 variables used to determine the values of locale categories.)

101167 *LC_ALL* If set to a non-empty string value, override the values of all the other
 101168 internationalization variables.

- 101169 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101170 characters (for example, single-byte as opposed to multi-byte characters in
 101171 arguments).
- 101172 *LC_MESSAGES*
 101173 Determine the locale that should be used to affect the format and contents of
 101174 diagnostic messages written to standard error.
- 101175 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 101176 *PATH* Determine the search path that is used to locate the utility to be invoked. See XBD
 101177 [Chapter 8](#) (on page 173).
- 101178 **ASYNCHRONOUS EVENTS**
- 101179 The *nohup* utility shall take the standard action for all signals except that *SIGHUP* shall be
 101180 ignored.
- 101181 **STDOUT**
- 101182 If the standard output is not a terminal, the standard output of *nohup* shall be the standard
 101183 output generated by the execution of the *utility* specified by the operands. Otherwise, nothing
 101184 shall be written to the standard output.
- 101185 **STDERR**
- 101186 If the standard output is a terminal, a message shall be written to the standard error, indicating
 101187 the name of the file to which the output is being appended. The name of the file shall be either
 101188 **nohup.out** or **\$HOME/nohup.out**.
- 101189 **OUTPUT FILES**
- 101190 Output written by the named utility is appended to the file **nohup.out** (or **\$HOME/nohup.out**),
 101191 if the conditions hold as described in the *DESCRIPTION*.
- 101192 **EXTENDED DESCRIPTION**
- 101193 None.
- 101194 **EXIT STATUS**
- 101195 The following exit values shall be returned:
- 101196 126 The utility specified by *utility* was found but could not be invoked.
- 101197 127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be
 101198 found.
- 101199 Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.
- 101200 **CONSEQUENCES OF ERRORS**
- 101201 Default.
- 101202 **APPLICATION USAGE**
- 101203 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
 101204 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
 101205 utility exited with an error indication”. The value 127 was chosen because it is not commonly
 101206 used for other meanings; most utilities use small values for “normal error conditions” and the
 101207 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
 101208 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
 101209 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
 101210 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
 101211 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
 101212 any other reason.

101213 EXAMPLES

101214 It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by
101215 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and
101216 the *nohup* applies to everything in the file.

101217 Alternatively, the following command can be used to apply *nohup* to a complex command:

```
101218 nohup sh -c 'complex-command-line' </dev/null
```

101219 RATIONALE

101220 The 4.3 BSD version ignores SIGTERM and SIGHUP, and if *./nohup.out* cannot be used, it fails
101221 instead of trying to use *\$HOME/nohup.out*.

101222 The *cs* utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this
101223 volume of POSIX.1-2017.

101224 The term *utility* is used, rather than *command*, to highlight the fact that shell compound
101225 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*
101226 includes user application programs and shell scripts, not just the standard utilities.

101227 Historical versions of the *nohup* utility use default file creation semantics. Some more recent
101228 versions use the permissions specified here as an added security precaution.

101229 Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore
101230 SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several
101231 reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this
101232 volume of POSIX.1-2017.

101233 Historical versions of *nohup* did not affect standard input, but that causes problems in the
101234 common scenario where the user logs into a system, types the command:

```
101235 nohup make &
```

101236 at the prompt, and then logs out. If standard input is not affected by *nohup*, the login session
101237 may not terminate for quite some time, since standard input remains open until *make* exits. To
101238 avoid this problem, POSIX.1-2017 allows implementations to redirect standard input if it is a
101239 terminal. Since the behavior is implementation-defined, portable applications that may run into
101240 the problem should redirect standard input themselves. For example, instead of:

```
101241 nohup make &
```

101242 an application can invoke:

```
101243 nohup make </dev/null &
```

101244 FUTURE DIRECTIONS

101245 None.

101246 SEE ALSO

101247 [Chapter 2](#) (on page 2345), *sh*

101248 [XBD Chapter 8](#) (on page 173)

101249 [XSH *signal\(\)*](#)

101250 CHANGE HISTORY

101251 First released in Issue 2.

101252 **Issue 7**
101253

Austin Group Interpretations 1003.1-2001 #104, #105, and #106 are applied.

101254 NAME

101255 od *†*'dump files in various formats

101256 SYNOPSIS

101257 od [-v] [-A *address_base*] [-j *skip*] [-N *count*] [-t *type_string*]...101258 [*file...*]101259 XSI od [-bcdosx] [*file*] [[+]offset[.][b]]

101260 DESCRIPTION

101261 The *od* utility shall write the contents of its input files to standard output in a user-specified
101262 format.

101263 OPTIONS

101264 The *od* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the order of
101265 XSI presentation of the **-t** options and the **-bcdosx** options is significant.

101266 The following options shall be supported:

101267 **-A** *address_base*101268 Specify the input offset base. See the EXTENDED DESCRIPTION section. The
101269 application shall ensure that the *address_base* option-argument is a character. The
101270 characters 'd', 'o', and 'x' specify that the offset base shall be written in
101271 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the
101272 offset shall not be written.101273 XSI **-b** Interpret bytes in octal. This shall be equivalent to **-t o1**.101274 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC_CTYPE*
101275 category. Certain non-graphic characters appear as C escapes: "NUL=\0",
101276 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal
101277 numbers.101278 XSI **-d** Interpret *words* (two-byte units) in unsigned decimal. This shall be equivalent to
101279 **-t u2**.101280 **-j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or
101281 seek past the first *skip* bytes in the concatenated input files. If the combined input is
101282 not at least *skip* bytes long, the *od* utility shall write a diagnostic message to
101283 standard error and exit with a non-zero exit status.101284 By default, the *skip* option-argument shall be interpreted as a decimal number.
101285 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;
101286 otherwise, with a leading '0', the offset shall be interpreted as an octal number.
101287 Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted
101288 as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is
101289 hexadecimal, any appended 'b' shall be considered to be the final hexadecimal
101290 digit.101291 **-N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as
101292 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a
101293 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an
101294 octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip* is
101295 specified) are not available, it shall not be considered an error; the *od* utility shall
101296 format the input that is available.

101297	XSI	-o	Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to -t o2 .
101298	XSI	-s	Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to
101299		-t d2 .	
101300		-t <i>type_string</i>	
101301			Specify one or more output types. See the EXTENDED DESCRIPTION section. The
101302			application shall ensure that the <i>type_string</i> option-argument is a string specifying
101303			the types to be used when writing the input data. The string shall consist of the
101304			type specification characters <i>a</i> , <i>c</i> , <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> , specifying named character,
101305			character, signed decimal, floating point, octal, unsigned decimal, and
101306			hexadecimal, respectively. The type specification characters <i>d</i> , <i>f</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be
101307			followed by an optional unsigned decimal integer that specifies the number of
101308			bytes to be transformed by each instance of the output type. The type specification
101309			character <i>f</i> can be followed by an optional <i>F</i> , <i>D</i> , or <i>L</i> indicating that the conversion
101310			should be applied to an item of type float , double , or long double , respectively.
101311			The type specification characters <i>d</i> , <i>o</i> , <i>u</i> , and <i>x</i> can be followed by an optional <i>C</i> , <i>S</i> ,
101312			<i>I</i> , or <i>L</i> indicating that the conversion should be applied to an item of type char ,
101313			short , int , or long , respectively. Multiple types can be concatenated within the
101314			same <i>type_string</i> and multiple -t options can be specified. Output lines shall be
101315			written for each type specified in the order in which the type specification
101316			characters are specified.
101317		-v	Write all input data. Without the -v option, any number of groups of output lines,
101318			which would be identical to the immediately preceding group of output lines
101319			(except for the byte offsets), shall be replaced with a line containing only an
101320			<asterisk> ('*').
101321	XSI	-x	Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to -t x2 .
101322	XSI		Multiple types can be specified by using multiple -bcdostx options. Output lines are written for
101323			each type specified in the order in which the types are specified.
101324		OPERANDS	
101325			The following operands shall be supported:
101326		<i>file</i>	A pathname of a file to be read. If no <i>file</i> operands are specified, the standard input
101327			shall be used.
101328			If there are no more than two operands, none of the -A , -j , -N , -t , or -v options is
101329			specified, and either of the following is true: the first character of the last operand
101330			is a <plus-sign> ('+'), or there are two operands and the first character of the last
101331	XSI		operand is numeric; the last operand shall be interpreted as an offset operand on
101332			XSI-conformant systems. Under these conditions, the results are unspecified on
101333			systems that are not XSI-conformant systems.
101334	XSI	[+]<i>offset</i>.[]<i>b</i>	The <i>offset</i> operand specifies the offset in the file where dumping is to commence.
101335			This operand is normally interpreted as octal bytes. If '.' is appended, the offset
101336			shall be interpreted in decimal. If 'b' is appended, the offset shall be interpreted
101337			in units of 512 bytes.
101338		STDIN	
101339			The standard input shall be used if no <i>file</i> operands are specified, and shall be used if a <i>file</i>
101340			operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
101341			the standard input shall not be used. See the INPUT FILES section.

101342 **INPUT FILES**

101343 The input files can be any file type.

101344 **ENVIRONMENT VARIABLES**

101345 The following environment variables shall affect the execution of *od*:

101346 *LANG* Provide a default value for the internationalization variables that are unset or null.
 101347 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 101348 variables used to determine the values of locale categories.)

101349 *LC_ALL* If set to a non-empty string value, override the values of all the other
 101350 internationalization variables.

101351 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101352 characters (for example, single-byte as opposed to multi-byte characters in
 101353 arguments and input files).

101354 *LC_MESSAGES*

101355 Determine the locale that should be used to affect the format and contents of
 101356 diagnostic messages written to standard error.

101357 *LC_NUMERIC*

101358 Determine the locale for selecting the radix character used when writing floating-
 101359 point formatted output.

101360 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

101361 **ASYNCHRONOUS EVENTS**

101362 Default.

101363 **STDOUT**

101364 See the EXTENDED DESCRIPTION section.

101365 **STDERR**

101366 The standard error shall be used only for diagnostic messages.

101367 **OUTPUT FILES**

101368 None.

101369 **EXTENDED DESCRIPTION**

101370 The *od* utility shall copy sequentially each input file to standard output, transforming the input
 101371 XSI data according to the output types specified by the *-t* option or the *-bcdosx* options. If no
 101372 output type is specified, the default output shall be as if *-t oS* had been specified.

101373 The number of bytes transformed by the output type specifier *c* may be variable depending on
 101374 the *LC_CTYPE* category.

101375 The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds
 101376 to the various C-language types as follows. If the *c99* compiler is present on the system, these
 101377 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes
 101378 may vary among systems that conform to POSIX.1-2017.

101379 For the type specifier characters *d*, *o*, *u*, and *x*, the default number of bytes shall
 101380 correspond to the size of the underlying implementation's basic integer type. For these
 101381 specifier characters, the implementation shall support values of the optional number of
 101382 bytes to be converted corresponding to the number of bytes in the C-language types **char**,
 101383 **short**, **int**, and **long**. These numbers can also be specified by an application as the
 101384 characters 'C', 'S', 'I', and 'L', respectively. The implementation shall also support
 101385 the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The

101386 implementation shall support the decimal value corresponding to the C-language type
 101387 **long long**. The byte order used when interpreting numeric values is implementation-
 101388 defined, but shall correspond to the order in which a constant of the corresponding type is
 101389 stored in memory on the system.

101390 For the type specifier character \mathfrak{f} , the default number of bytes shall correspond to the
 101391 number of bytes in the underlying implementation's basic double precision floating-point
 101392 data type. The implementation shall support values of the optional number of bytes to be
 101393 converted corresponding to the number of bytes in the C-language types **float**, **double**,
 101394 and **long double**. These numbers can also be specified by an application as the characters
 101395 'F', 'D', and 'L', respectively.

101396 The type specifier character \mathfrak{a} specifies that bytes shall be interpreted as named characters from
 101397 the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least
 101398 significant seven bits of each byte shall be used for this type specification. Bytes with the values
 101399 listed in the following table shall be written using the corresponding names for those characters.

101400 **Table 4-13** Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

101411 **Note:** The "\012" value may be written either as **lf** or **nl**.

101412 The type specifier character \mathfrak{c} specifies that bytes shall be interpreted as characters specified by
 101413 the current setting of the *LC_CTYPE* locale category. Characters listed in the table in XBD
 101414 [Chapter 5](#) (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be written as
 101415 the corresponding escape sequences, except that <backslash> shall be written as a single
 101416 <backslash> and a NUL shall be written as '\0'. Other non-printable characters shall be
 101417 written as one three-digit octal number for each byte in the character. Printable multi-byte
 101418 characters shall be written in the area corresponding to the first byte of the character; the two-
 101419 character sequence "***" shall be written in the area corresponding to each remaining byte in the
 101420 character, as an indication that the character is continued. When either the **-j skip** or **-N count**
 101421 option is specified along with the \mathfrak{c} type specifier, and this results in an attempt to start or finish
 101422 in the middle of a multi-byte character, the result is implementation-defined.

101423 The input data shall be manipulated in blocks, where a block is defined as a multiple of the least
 101424 common multiple of the number of bytes transformed by the specified output types. If the least
 101425 common multiple is greater than 16, the results are unspecified. Each input block shall be
 101426 written as transformed by each output type, one per written line, in the order that the output
 101427 types were specified. If the input block size is larger than the number of bytes transformed by
 101428 the output type, the output type shall sequentially transform the parts of the input block, and
 101429 the output from each of the transformations shall be separated by one or more <blank>
 101430 characters.

101431 If, as a result of the specification of the **-N** option or end-of-file being reached on the last input
 101432 file, input data only partially satisfies an output type, the input shall be extended sufficiently

101433 with null bytes to write the last byte of the input.

101434 Unless **-A n** is specified, the first output line produced for each input block shall be preceded by
 101435 the input offset, cumulative across input files, of the next byte to be written. The format of the
 101436 input offset is unspecified; however, it shall not contain any <blank> characters, shall start at the
 101437 first character of the output line, and shall be followed by one or more <blank> characters. In
 101438 addition, the offset of the byte following the last byte written shall be written after all the input
 101439 data has been processed, but shall not be followed by any <blank> characters.

101440 If no **-A** option is specified, the input offset base is unspecified.

101441 EXIT STATUS

101442 The following exit values shall be returned:

101443 0 All input files were processed successfully.

101444 >0 An error occurred.

101445 CONSEQUENCES OF ERRORS

101446 Default.

101447 APPLICATION USAGE

101448 XSI-conformant applications are warned not to use filenames starting with '+' or a first
 101449 operand starting with a numeric character so that the old functionality can be maintained by
 101450 implementations, unless they specify one of the **-A**, **-j**, or **-N** options. To guarantee that one of
 101451 these filenames is always interpreted as a filename, an application could always specify the
 101452 address base format with the **-A** option.

101453 EXAMPLES

101454 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as
 101455 standard input to the command:

```
101456 od -A d -t a
```

101457 on an implementation using an input block size of 16 bytes, the standard output, independent of
 101458 the current locale setting, would be similar to:

```
101459 0000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
101460 0000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
101461 0000032 sp ! " # $ % & ' ( ) * + , - . /
101462 0000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
101463 0000064 @ A B C D E F G H I J K L M N O
101464 0000080 P Q R S T U V W X Y Z [ \ ] ^ _
101465 0000096 ` a b c d e f g h i j k l m n o
101466 0000112 p q r s t u v w x y z { | } ~ del
101467 0000128
```

101468 Note that this volume of POSIX.1-2017 allows **nl** or **lf** to be used as the name for the
 101469 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character
 101470 **lf** (line feed), but traditional implementations have referred to this character as **newline** (**nl**) and
 101471 the POSIX locale character set symbolic name for the corresponding character is a <newline>.

101472 The command:

```
101473 od -A o -t o2x2x -N 18
```

101474 on a system with 32-bit words and an implementation using an input block size of 16 bytes
 101475 could write 18 bytes in approximately the following format:

```
101476 0000000 032056 031440 041123 042040 052516 044530 020043 031464
```

```

101477          342e   3320   4253   4420   554e   4958   2023   3334
101478          342e3320         42534420         554e4958         20233334
101479 0000020 032472
101480          353a
101481          353a0000
101482 0000022

```

101483 **The command:**

```
101484 od -A d -t f -t o4 -t x4 -N 24 -j 0x15
```

101485 on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point
101486 format) would skip 21 bytes of input data and then write 24 bytes in approximately the
101487 following format:

```

101488 0000000 1.0000000000000000e+00 1.5735000000000000e+01
101489 07774000000 00000000000 10013674121 35341217270
101490 3ff00000 00000000 402f3851 eb851eb8
101491 0000016 1.4066823000000000e+02
101492 10030312542 04370303230
101493 40619562 23e18698
101494 0000024

```

101495 **RATIONALE**

101496 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently
101497 *hexdump*. There were several objections to all of these based on the following reasons:

101498 The *hd* and *xd* names conflicted with historical utilities that behaved differently.

101499 The *hexdump* description was much more complex than needed for a simple dump utility.

101500 The *od* utility has been available on all historical implementations and there was no need to
101501 create a new name for a utility so similar to the historical *od* utility.

101502 The original reasons for not standardizing historical *od* were also fairly widespread. Those
101503 reasons are given below along with rationale explaining why the standard developers believe
101504 that this version does not suffer from the indicated problem:

101505 The BSD and System V versions of *od* have diverged, and the intersection of features
101506 provided by both does not meet the needs of the user community. In fact, the System V
101507 version only provides a mechanism for dumping octal bytes and **shorts**, signed and
101508 unsigned decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability
101509 to dump **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned
101510 decimal, and hexadecimal **longs**. The version presented here provides more normalized
101511 forms for dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned
101512 decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as
101513 current locale characters.

101514 It would not be possible to come up with a compatible superset of the BSD and System V
101515 flags that met the requirements of the standard developers. The historical default *od* output
101516 is the specified default output of this utility. None of the option letters chosen for this
101517 version of *od* conflict with any of the options to historical versions of *od*.

101518 On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps
101519 of **ints**, even in the BSD version. Because of the way options are named, the name space
101520 could not be extended to solve these problems. This is why the **-t** option was added (with
101521 type specifiers more closely matched to the *printf()* formats used in the rest of this volume
101522 of POSIX.1-2017) and the optional field sizes were added to the **d**, **f**, **o**, **u**, and **x** type

101523 specifiers. It is also one of the reasons why the historical practice was not mandated as a
 101524 required obsolescent form of *od*. (Although the old versions of *od* are not listed as an
 101525 obsolescent form, implementations are urged to continue to recognize the older forms for
 101526 several more years.) The *a*, *c*, *f*, *o*, and *x* types match the meaning of the corresponding
 101527 format characters in the historical implementations of *od* except for the default sizes of the
 101528 fields converted. The *d* format is signed in this volume of POSIX.1-2017 to match the
 101529 *printf()* notation. (Historical versions of *od* used *d* as a synonym for *u* in this version. The
 101530 System V implementation uses *s* for signed decimal; BSD uses *i* for signed decimal and *s*
 101531 for null-terminated strings.) Other than *d* and *u*, all of the type specifiers match format
 101532 characters in the historical BSD version of *od*.

101533 The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are
 101534 used even though it is recognized that there may be zero or more than one compiler for the
 101535 C language on an implementation and that they may use different sizes for some of these
 101536 types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**,
 101537 while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes
 101538 **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the
 101539 implementation for these types, corresponding to the values reported by invocations of the
 101540 *getconf* utility when called with *system_var* operands {UCHAR_MAX}, {USHORT_MAX},
 101541 {UINT_MAX}, and {ULONG_MAX} for the types **char**, **short**, **int**, and **long**, respectively.
 101542 There are similar constants required by the ISO C standard, but not required by the System
 101543 Interfaces volume of POSIX.1-2017 or this volume of POSIX.1-2017. They are
 101544 {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types **float**,
 101545 **double**, and **long double**, respectively. If the optional *c99* utility is provided by the
 101546 implementation and used as specified by this volume of POSIX.1-2017, these are the sizes
 101547 that would be provided. If an option is used that specifies different sizes for these types,
 101548 there is no guarantee that the *od* utility is able to interpret binary data output by such a
 101549 program correctly.

101550 This volume of POSIX.1-2017 requires that the numeric values of these lengths be
 101551 recognized by the *od* utility and that symbolic forms also be recognized. Thus, a
 101552 conforming application can always look at an array of **unsigned long** data elements using
 101553 *od -t uL*.

101554 The method of specifying the format for the address field based on specifying a starting
 101555 offset in a file unnecessarily tied the two together. The **-A** option now specifies the address
 101556 base and the **-S** option specifies a starting offset.

101557 It would be difficult to break the dependence on US ASCII to achieve an internationalized
 101558 utility. It does not seem to be any harder for *od* to dump characters in the current locale
 101559 than it is for the *ed* or *sed* **I** commands. The *c* type specifier does this without difficulty and
 101560 is completely compatible with the historical implementations of the *c* format character
 101561 when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The
 101562 *a* type specifier (from the BSD **a** format character) was left as a portable means to dump
 101563 ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by
 101564 *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard
 101565 as a subset of their base codeset.

101566 The use of "***" as an indication of continuation of a multi-byte character in *c* specifier output
 101567 was chosen based on seeing an implementation that uses this method. The continuation bytes
 101568 have to be marked in a way that is not ambiguous with another single-byte or multi-byte
 101569 character.

101570 An early proposal used **-S** and **-n**, respectively, for the **-j** and **-N** options eventually selected.
 101571 These were changed to avoid conflicts with historical implementations.

101572 The original standard specified `-t o2` as the default when no output type was given. This was
101573 changed to `-t oS` (the length of a **short**) to accommodate a supercomputer implementation that
101574 historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not
101575 affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at
101576 the same time to address an historical implementation that had no two-byte data types in its C
101577 compiler.

101578 The use of a basic integer data type is intended to allow the implementation to choose a word
101579 size commonly used by applications on that architecture.

101580 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
101581 but this has been modified in this version.

101582 **FUTURE DIRECTIONS**

101583 All option and operand interfaces marked XSI may be removed in a future version.

101584 **SEE ALSO**

101585 *c99*, *sed*

101586 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

101587 **CHANGE HISTORY**

101588 First released in Issue 2.

101589 **Issue 5**

101590 In the description of the `-c` option, the phrase “This is equivalent to `-t c.`” is deleted.

101591 The FUTURE DIRECTIONS section is modified.

101592 **Issue 6**

101593 The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the
101594 revisions in the IEEE P1003.2b draft standard.

101595 The normative text is reworded to avoid use of the term “must” for application requirements.

101596 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples
101597 which used an undefined `-n` option, which should have been `-N`.

101598 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/19 is applied, removing text describing
101599 behavior on systems with bytes consisting of more than eight bits.

101600 **Issue 7**

101601 Austin Group Interpretation 1003.1-2001 #092 is applied.

101602 SD5-XCU-ERN-37 is applied, updating the OPERANDS section.

101603 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

101604 **NAME**101605 `paste` — merge corresponding or subsequent lines of files101606 **SYNOPSIS**101607 `paste [-s] [-d list] file...`101608 **DESCRIPTION**101609 The *paste* utility shall concatenate the corresponding lines of the given input files, and write the
101610 resulting lines to standard output.101611 The default operation of *paste* shall concatenate the corresponding lines of the input files. The
101612 <newline> of every line except the line from the last input file shall be replaced with a <tab>.101613 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall
101614 behave as though empty lines were read from the files on which end-of-file was detected, unless
101615 the `-s` option is specified.101616 **OPTIONS**101617 The *paste* utility shall conform to XBD [Section 12.2](#) (on page 216).

101618 The following options shall be supported:

101619 `-d list` Unless a <backslash> character appears in *list*, each character in *list* is an element
101620 specifying a delimiter character. If a <backslash> character appears in *list*, the
101621 <backslash> character and one or more characters following it are an element
101622 specifying a delimiter character as described below. These elements specify one or
101623 more delimiters to use, instead of the default <tab>, to replace the <newline> of
101624 the input lines. The elements in *list* shall be used circularly; that is, when the list is
101625 exhausted the first element from the list is reused. When the `-s` option is specified:

101626 The last <newline> in a file shall not be modified.

101627 The delimiter shall be reset to the first element of *list* after each *file* operand is
101628 processed.101629 When the `-s` option is not specified:101630 The <newline> characters in the file specified by the last *file* operand shall
101631 not be modified.101632 The delimiter shall be reset to the first element of *list* each time a line is
101633 processed from each file.101634 If a <backslash> character appears in *list*, it and the character following it shall be
101635 used to represent the following delimiter characters:101636 `\n` <newline>.101637 `\t` <tab>.101638 `\\` <backslash> character.101639 `\0` Empty string (not a null character). If `'\0'` is immediately followed by the
101640 character `'x'`, the character `'X'`, or any character defined by the `LC_CTYPE`
101641 **digit** keyword (see XBD [Chapter 7](#), on page 135), the results are unspecified.

101642 If any other characters follow the <backslash>, the results are unspecified.

101643 `-s` Concatenate all of the lines from each input file into one line of output per file, in
101644 command line order. The <newline> of every line except the last line in each input
101645 file shall be replaced with a <tab>, unless otherwise specified by the `-d` option. If
101646 an input file is empty, the output line corresponding to that file shall consist of

101647 only a <newline> character.

101648 **OPERANDS**

101649 The following operand shall be supported:

101650 *file* A pathname of an input file. If '-' is specified for one or more of the *files*, the
 101651 standard input shall be used; the standard input shall be read one line at a time,
 101652 circularly, for each instance of '-'. Implementations shall support pasting of at
 101653 least 12 *file* operands.

101654 **STDIN**

101655 The standard input shall be used only if one or more *file* operands is '-'. See the INPUT FILES
 101656 section.

101657 **INPUT FILES**

101658 The input files shall be text files, except that line lengths shall be unlimited.

101659 **ENVIRONMENT VARIABLES**

101660 The following environment variables shall affect the execution of *paste*:

101661 *LANG* Provide a default value for the internationalization variables that are unset or null.
 101662 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
 101663 used to determine the values of locale categories.)

101664 *LC_ALL* If set to a non-empty string value, override the values of all the other
 101665 internationalization variables.

101666 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101667 characters (for example, single-byte as opposed to multi-byte characters in
 101668 arguments and input files).

101669 *LC_MESSAGES*

101670 Determine the locale that should be used to affect the format and contents of
 101671 diagnostic messages written to standard error.

101672 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

101673 **ASYNCHRONOUS EVENTS**

101674 Default.

101675 **STDOUT**

101676 Concatenated lines of input files shall be separated by the <tab> (or other characters under the
 101677 control of the *-d* option) and terminated by a <newline>.

101678 **STDERR**

101679 The standard error shall be used only for diagnostic messages.

101680 **OUTPUT FILES**

101681 None.

101682 **EXTENDED DESCRIPTION**

101683 None.

101684 **EXIT STATUS**

101685 The following exit values shall be returned:

101686 0 Successful completion.

101687 >0 An error occurred.

101688 **CONSEQUENCES OF ERRORS**

101689 If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic
 101690 message shall be written to standard error, but no output is written to standard output. If the `-s`
 101691 option is specified, the *paste* utility shall provide the default behavior described in [Section 1.4](#) (on
 101692 page 2336).

101693 **APPLICATION USAGE**

101694 When the escape sequences of the *list* option-argument are used in a shell script, they must be
 101695 quoted; otherwise, the shell treats the `<backslash>` as a special character.

101696 Conforming applications should only use the specific `<backslash>`-escaped delimiters presented
 101697 in this volume of POSIX.1-2017. Historical implementations treat `'\x'`, where `'x'` is not in this
 101698 list, as `'x'`, but future implementations are free to expand this list to recognize other common
 101699 escapes similar to those accepted by *printf* and other standard utilities.

101700 Most of the standard utilities work on text files. The *cut* utility can be used to turn files with
 101701 arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used
 101702 to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
101703 cut -b 1-500 -n file > file1
101704 cut -b 501- -n file > file2
```

101705 creates **file1** (a text file) with lines no longer than 500 bytes (plus the `<newline>`) and **file2** that
 101706 contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in
 101707 *file* that are longer than `500 + {LINE_MAX}` bytes. The original file can be recreated from **file1**
 101708 and **file2** using the command:

```
101709 paste -d "\0" file1 file2 > file
```

101710 The commands:

```
101711 paste -d "\0" ...
101712 paste -d "" ...
```

101713 are not necessarily equivalent; the latter is not specified by this volume of POSIX.1-2017 and
 101714 may result in an error. The construct `'\0'` is used to mean “no separator” because historical
 101715 versions of *paste* did not follow the syntax guidelines, and the command:

```
101716 paste -d"" ...
```

101717 could not be handled properly by *getopt()*.

101718 **EXAMPLES**

101719 1. Write out a directory in four columns:

```
101720 ls | paste - - - -
```

101721 2. Combine pairs of lines from a file into single lines:

```
101722 paste -s -d "\t\n" file
```

101723 **RATIONALE**

101724 None.

101725 **FUTURE DIRECTIONS**

101726 None.

101727 **SEE ALSO**101728 [Section 1.4](#) (on page 2336), *cut*, *grep*, *pr*101729 XBD [Chapter 7](#) (on page 135), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)101730 **CHANGE HISTORY**

101731 First released in Issue 2.

101732 **Issue 6**

101733 The normative text is reworded to avoid use of the term “must” for application requirements.

101734 **Issue 7**

101735 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

101736 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0149 [973] is applied.

101737 **NAME**

101738 patch ‡'apply changes to files

101739 **SYNOPSIS**101740 patch [-blNR] [-c|-e|-n|-u] [-d *dir*] [-D *define*] [-i *patchfile*]
101741 [-o *outfile*] [-p *num*] [-r *rejectfile*] [*file*]101742 **DESCRIPTION**101743 The *patch* utility shall read a source (patch) file containing any of four forms of difference (diff)
101744 listings produced by the *diff* utility (normal, copied context, unified context, or in the style of *ed*)
101745 and apply those differences to a file. By default, *patch* shall read from the standard input.101746 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a *-c*,
101747 *-e*, *-n*, or *-u* option.101748 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they
101749 came from separate patch files. (In this case, the application shall ensure that the name of the
101750 patch file is determinable for each *diff* listing.)101751 **OPTIONS**101752 The *patch* utility shall conform to XBD [Section 12.2](#) (on page 216).

101753 The following options shall be supported:

101754 **-b** Save a copy of the original contents of each modified file, before the differences are
101755 applied, in a file of the same name with the suffix **.orig** appended to it. If the file
101756 already exists, it shall be overwritten; if multiple patches are applied to the same
101757 file, the **.orig** file shall be written only for the first patch. When the *-o outfile* option
101758 is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig*
101759 shall be created.101760 **-c** Interpret the patch file as a copied context difference (the output of the utility *diff*
101761 when the *-c* or *-C* options are specified).101762 **-d dir** Change the current directory to *dir* before processing as described in the
101763 EXTENDED DESCRIPTION section.101764 **-D define** Mark changes with one of the following C preprocessor constructs:101765 #ifdef define
101766 ...
101767 #endif

101768 #ifndef define
101769 ...
101770 #endif101771 optionally combined with the C preprocessor construct **#else**. If the patched file is
101772 processed with the C preprocessor, where the macro *define* is defined, the output
101773 shall contain the changes from the patch file; otherwise, the output shall not
101774 contain the patches specified in the patch file.101775 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.101776 **-i patchfile** Read the patch information from the file named by the pathname *patchfile*, rather
101777 than the standard input.101778 **-l** (The letter ell.) Cause any sequence of <blank> characters in the difference script to
101779 match any sequence of <blank> characters in the input file. Other characters shall
101780 be matched exactly.

- 101781 **-n** Interpret the script as a normal difference.
- 101782 **-N** Ignore patches where the differences have already been applied to the file; by
101783 default, already-applied patches shall be rejected.
- 101784 **-o *outfile*** Instead of modifying the files (specified by the *file* operand or the difference
101785 listings) directly, write a copy of the file referenced by each patch, with the
101786 appropriate differences applied, to *outfile*. Multiple patches for a single file shall be
101787 applied to the intermediate versions of the file created by any previous patches,
101788 and shall result in multiple, concatenated versions of the file being written to
101789 *outfile*.
- 101790 **-p *num*** For all pathnames in the patch file that indicate the names of files to be patched,
101791 delete *num* pathname components from the beginning of each pathname. If the
101792 pathname in the patch file is absolute, any leading <slash> characters shall be
101793 considered the first component (that is, **-p 1** shall remove the leading <slash>
101794 characters). Specifying **-p 0** shall cause the full pathname to be used. If **-p** is not
101795 specified, only the basename (the final pathname component) shall be used.
- 101796 **-R** Reverse the sense of the patch script; that is, assume that the difference script was
101797 created from the new version to the old version. The **-R** option cannot be used
101798 with *ed* scripts. The *patch* utility shall attempt to reverse each portion of the script
101799 before applying it. Rejected differences shall be saved in swapped format. If this
101800 option is not specified, and until a portion of the patch file is successfully applied,
101801 *patch* attempts to apply each portion in its reversed sense as well as in its normal
101802 sense. If the attempt is successful, the user shall be prompted to determine whether
101803 the **-R** option should be set.
- 101804 **-r *rejectfile*** Override the default reject filename. In the default case, the reject file shall have the
101805 same name as the output file, with the suffix **.rej** appended to it; see [Patch](#)
101806 [Application](#) (on page 3059).
- 101807 **-u** Interpret the patch file as a unified context difference (the output of the *diff* utility
101808 when the **-u** or **-U** options are specified).

101809 OPERANDS

101810 The following operand shall be supported:

101811 *file* A pathname of a file to patch.

101812 STDIN

101813 See the INPUT FILES section.

101814 INPUT FILES

101815 Input files shall be text files.

101816 ENVIRONMENT VARIABLES

101817 The following environment variables shall affect the execution of *patch*:

101818 **LANG** Provide a default value for the internationalization variables that are unset or null.
101819 (See [XBD Section 8.2](#) (on page 174) the precedence of internationalization variables
101820 used to determine the values of locale categories.)

101821 **LC_ALL** If set to a non-empty string value, override the values of all the other
101822 internationalization variables.

101823 **LC_COLLATE**

101824 Determine the locale for the behavior of ranges, equivalence classes, and multi-
101825 character collating elements used in the extended regular expression defined for

101826 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

101827 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101828 characters (for example, single-byte as opposed to multi-byte characters in
 101829 arguments and input files), and the behavior of character classes used in the
 101830 extended regular expression defined for the **yesexpr** locale keyword in the
 101831 *LC_MESSAGES* category.

101832 *LC_MESSAGES*
 101833 Determine the locale used to process affirmative responses, and the locale used to
 101834 affect the format and contents of diagnostic messages and prompts written to
 101835 standard error.

101836 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

101837 *LC_TIME* Determine the locale for recognizing the format of file timestamps written by the
 101838 *diff* utility in a context-difference input file.

101839 **ASYNCHRONOUS EVENTS**

101840 Default.

101841 **STDOUT**

101842 Not used.

101843 **STDERR**

101844 The standard error shall be used for diagnostic and informational messages.

101845 **OUTPUT FILES**

101846 The output of the *patch* utility, the save files (**.orig** suffixes), and the reject files (**.rej** suffixes) shall
 101847 be text files.

101848 **EXTENDED DESCRIPTION**

101849 A patch file may contain patching instructions for more than one file; filenames shall be
 101850 determined as specified in [Filename Determination](#) (on page 3059). When the **-b** option is
 101851 specified, for each patched file, the original shall be saved in a file of the same name with the
 101852 suffix **.orig** appended to it.

101853 For each patched file, a reject file may also be created as noted in [Patch Application](#) (on page
 101854 3059). In the absence of a **-r** option, the name of this file shall be formed by appending the suffix
 101855 **.rej** to the original filename.

101856 **Patch File Format**

101857 The patch file shall contain zero or more lines of header information followed by one or more
 101858 patches. Each patch shall contain zero or more lines of filename identification in the format
 101859 produced by the **-c**, **-C**, **-u**, or **-U** options of the *diff* utility, and one or more sets of *diff* output,
 101860 which are customarily called *hunks*.

101861 The *patch* utility shall recognize the following expression in the header information:

101862 **Index:** *pathname*
 101863 The file to be patched is named *pathname*.

101864 If all lines (including headers) within a patch begin with the same leading sequence of <blank>
 101865 characters, the *patch* utility shall remove this sequence before proceeding. Within each patch, if
 101866 the type of difference is common context, the *patch* utility shall recognize the following
 101867 expressions:

101868 *** *filename timestamp*
 101869 The patches arose from *filename*.

101870 --- *filename timestamp*
 101871 The patches should be applied to *filename*.

101872 If the type of difference is unified context, the *patch* utility shall recognize the following
 101873 expressions:

101874 --- *filename timestamp*
 101875 The patches arose from *filename*.

101876 + + + *filename timestamp*
 101877 The patches should be applied to *filename*.

101878 Each hunk within a patch shall be the *diff* output to change a line range within the original file.
 101879 The line numbers for successive hunks within a patch shall occur in ascending order.

101880 **Filename Determination**

101881 If no *file* operand is specified, *patch* shall perform the following steps to determine the filename
 101882 to use:

101883 1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as
 101884 specified by the **-p** option) from the filename on the line beginning with "****" (if copied
 101885 context) or "---" (if unified context), then test for the existence of this file relative to the
 101886 current directory (or the directory specified with the **-d** option). If the file exists, the *patch*
 101887 utility shall use this filename.

101888 2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as
 101889 specified by the **-p** option) from the filename on the line beginning with "---" (if copied
 101890 context) or "+ + +" (if unified context), then test for the existence of this file relative to the
 101891 current directory (or the directory specified with the **-d** option). If the file exists, the *patch*
 101892 utility shall use this filename.

101893 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility
 101894 shall delete pathname components (as specified by the **-p** option) from this line, then test
 101895 for the existence of this file relative to the current directory (or the directory specified
 101896 with the **-d** option). If the file exists, the *patch* utility shall use this filename.

101897 XSI 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a `get -e`
 101898 `SCCS/s.filename` command to retrieve an editable version of the file. If the file exists, the
 101899 *patch* utility shall use this filename.

101900 5. The *patch* utility shall write a prompt to standard output and request a filename
 101901 interactively from the controlling terminal (for example, **/dev/tty**).

101902 **Patch Application**

101903 If the **-c**, **-e**, **-n**, or **-u** option is present, the *patch* utility shall interpret information within each
 101904 hunk as a copied context difference, an *ed* difference, a normal difference, or a unified context
 101905 difference, respectively. In the absence of any of these options, the *patch* utility shall determine
 101906 the type of difference based on the format of information within the hunk.

101907 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line
 101908 number at the beginning of the hunk, plus or minus any offset used in applying the previous
 101909 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and
 101910 backwards at least 1 000 bytes for a set of lines that match the hunk context.

101911 If no such place is found and it is a context difference, then another scan shall take place,
 101912 ignoring the first and last line of context. If that fails, the first two and last two lines of context
 101913 shall be ignored and another scan shall be made. Implementations may search more extensively
 101914 for installation locations.

101915 If no location can be found, the *patch* utility shall append the hunk to the reject file. A rejected
 101916 hunk that is a copied context difference, an *ed* difference, or a normal difference shall be written
 101917 in copied-context-difference format regardless of the format of the patch file. It is
 101918 implementation-defined whether a rejected hunk that is a unified context difference is written in
 101919 copied-context-difference format or in unified-context-difference format. If the input was a
 101920 normal or *ed*-style difference, the reject file may contain differences with zero lines of context.
 101921 The line numbers on the hunks in the reject file may be different from the line numbers in the
 101922 patch file since they shall reflect the approximate locations for the failed hunks in the new file
 101923 rather than the old one.

101924 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking
 101925 the *ed* utility.

101926 EXIT STATUS

101927 The following exit values shall be returned:

- 101928 0 Successful completion.
- 101929 1 One or more lines were written to a reject file.
- 101930 >1 An error occurred.

101931 CONSEQUENCES OF ERRORS

101932 Patches that cannot be correctly placed in the file shall be written to a reject file.

101933 APPLICATION USAGE

101934 The **-R** option does not work with *ed* scripts because there is too little information to reconstruct
 101935 the reverse operation.

101936 The **-p** option makes it possible to customize a patch file to local user directory structures
 101937 without manually editing the patch file. For example, if the filename in the patch file was:

101938 `/curds/whey/src/blurfl/blurfl.c`

101939 Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

101940 `curds/whey/src/blurfl/blurfl.c`

101941 without the leading `<slash>`, **-p 4** gives:

101942 `blurfl/blurfl.c`

101943 and not specifying **-p** at all gives:

101944 `blurfl.c .`

101945 EXAMPLES

101946 None.

101947 RATIONALE

101948 Some of the functionality in historical *patch* implementations was not specified. The following
 101949 documents those features present in historical implementations that have not been specified.

101950 A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options
 101951 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

101952 In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility

101953 would search for the corresponding version information (the string specified in the header,
101954 delimited by <blank> characters or the beginning or end of a line or the file) anywhere in the
101955 original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to
101956 standardize. For example, if:

101957 Prereq: 1.2

101958 were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the
101959 prerequisite.

101960 The following options were dropped from historical implementations of *patch* as insufficiently
101961 useful to standardize:

101962 **-b** The **-b** option historically provided a method for changing the name extension of
101963 the backup file from the default **.orig**. This option has been modified and retained
101964 in this volume of POSIX.1-2017.

101965 **-F** The **-F** option specified the number of lines of a context diff to ignore when
101966 searching for a place to install a patch.

101967 **-f** The **-f** option historically caused *patch* not to request additional information from
101968 the user.

101969 **-r** The **-r** option historically provided a method of overriding the extension of the
101970 reject file from the default **.rej**.

101971 **-s** The **-s** option historically caused *patch* to work silently unless an error occurred.

101972 **-x** The **-x** option historically set internal debugging flags.

101973 In some file system implementations, the saving of a **.orig** file may produce unwanted results. In
101974 the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum
101975 filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename
101976 limit. It was suggested, due to some historical practice, that a <tilde> ('~') suffix be used
101977 instead of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is
101978 not obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more
101979 understandable.

101980 The **-b** option has the opposite sense in some historical implementations ~~to~~ do not save the **orig**
101981 file. The default case here is not to save the files, making *patch* behave more consistently with the
101982 other standard utilities.

101983 The **-w** option in early proposals was changed to **-I** to match historical practice.

101984 The **-N** option was included because without it, a non-interactive application cannot reject
101985 previously applied patches. For example, if a user is piping the output of *diff* into the *patch*
101986 utility, and the user only wants to patch a file to a newer version non-interactively, the **-N** option
101987 is required.

101988 Changes to the **-I** option description were proposed to allow matching across <newline>
101989 characters in addition to just <blank> characters. Since this is not historical practice, and since
101990 some ambiguities could result, it is suggested that future developments in this area utilize
101991 another option letter, such as **-L**.

101992 The **-u** option of GNU *patch* has been added, along with support for unified context formats.

101993 **FUTURE DIRECTIONS**

101994 None.

101995 **SEE ALSO**101996 *diff, ed*101997 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)101998 **CHANGE HISTORY**

101999 First released in Issue 4.

102000 **Issue 5**

102001 The FUTURE DIRECTIONS section is added.

102002 **Issue 6**

102003 This utility is marked as part of the User Portability Utilities option.

102004 The description of the `-D` option and the steps in [Filename Determination](#) (on page 3059) are
102005 changed to match historical practice as defined in the IEEE P1003.2b draft standard.

102006 The normative text is reworded to avoid use of the term “must” for application requirements.

102007 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the
102008 *patch* utility performs `ifdef` selection for the `-D` option.102009 **Issue 7**102010 The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability
102011 Utilities is now an option for interactive utilities.

102012 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

102013 SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.102014 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
102015 `LC_MESSAGES` and `LC_CTYPE` environment variables and adding the `LC_COLLATE`
102016 environment variable.

102017 **NAME**

102018 pathchk ‡'check pathnames

102019 **SYNOPSIS**102020 pathchk [-p] [-P] *pathname...*102021 **DESCRIPTION**

102022 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used
 102023 to access or create a file without causing syntax errors) and portable (that is, no filename
 102024 truncation results). More extensive portability checks are provided by the **-p** and **-P** options.

102025 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the
 102026 underlying file system. A diagnostic shall be written for each *pathname* operand that:

102027 Is longer than {PATH_MAX} bytes (see **Pathname Variable Values** in XBD [<limits.h>](#))

102028 Contains any component longer than {NAME_MAX} bytes in its containing directory

102029 Contains any component in a directory that is not searchable

102030 Contains any byte sequence that is not valid in its containing directory

102031 The format of the diagnostic message is not specified, but shall indicate the error detected and
 102032 the corresponding *pathname* operand.

102033 It shall not be considered an error if one or more components of a *pathname* operand do not exist
 102034 as long as a file matching the pathname specified by the missing components could be created
 102035 that does not violate any of the checks specified above.

102036 **OPTIONS**

102037 The *pathchk* utility shall conform to XBD [Section 12.2](#) (on page 216).

102038 The following option shall be supported:

102039 **-p** Instead of performing checks based on the underlying file system, write a
 102040 diagnostic for each *pathname* operand that:

102041 Is longer than {_POSIX_PATH_MAX} bytes (see **Minimum Values** in XBD
 102042 [<limits.h>](#))

102043 Contains any component longer than {_POSIX_NAME_MAX} bytes

102044 Contains any character in any component that is not in the portable filename
 102045 character set

102046 **-P** Write a diagnostic for each *pathname* operand that:

102047 Contains a component whose first character is the <hyphen-minus>
 102048 character

102049 Is empty

102050 **OPERANDS**

102051 The following operand shall be supported:

102052 *pathname* A pathname to be checked.

102053 **STDIN**

102054 Not used.

102055 **INPUT FILES**

102056 None.

102057 **ENVIRONMENT VARIABLES**102058 The following environment variables shall affect the execution of *pathchk*:

102059 *LANG* Provide a default value for the internationalization variables that are unset or null.
 102060 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
 102061 used to determine the values of locale categories.)

102062 *LC_ALL* If set to a non-empty string value, override the values of all the other
 102063 internationalization variables.

102064 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 102065 characters (for example, single-byte as opposed to multi-byte characters in
 102066 arguments).

102067 *LC_MESSAGES*

102068 Determine the locale that should be used to affect the format and contents of
 102069 diagnostic messages written to standard error.

102070 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

102071 **ASYNCHRONOUS EVENTS**

102072 Default.

102073 **STDOUT**

102074 Not used.

102075 **STDERR**

102076 The standard error shall be used only for diagnostic messages.

102077 **OUTPUT FILES**

102078 None.

102079 **EXTENDED DESCRIPTION**

102080 None.

102081 **EXIT STATUS**

102082 The following exit values shall be returned:

102083 0 All *pathname* operands passed all of the checks.

102084 >0 An error occurred.

102085 **CONSEQUENCES OF ERRORS**

102086 Default.

102087 **APPLICATION USAGE**

102088 The *test* utility can be used to determine whether a given pathname names an existing file; it
 102089 does not, however, give any indication of whether or not any component of the pathname was
 102090 truncated in a directory where the `_POSIX_NO_TRUNC` feature is not in effect. The *pathchk*
 102091 utility does not check for file existence; it performs checks to determine whether a pathname
 102092 does exist or could be created with no pathname component truncation.

102093 The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a
 102094 file. As with all file creation semantics in the System Interfaces volume of POSIX.1-2017, it
 102095 guarantees atomic creation, but still depends on applications to agree on conventions and
 102096 cooperate on the use of files after they have been created.

102097 To verify that a pathname meets the requirements of filename portability, applications should

102098 use both the `-p` and `-P` options together.

102099 EXAMPLES

102100 To verify that all pathnames in an imported data interchange archive are legitimate and
102101 unambiguous on the current system:

```
102102 # This example assumes that no pathnames in the archive
102103 # contain <newline> characters.
102104 pax -f archive | sed -e 's/^[[:alnum:]]/\\&/g' | xargs pathchk --
102105 if [ $? -eq 0 ]
102106 then
102107     pax -r -f archive
102108 else
102109     echo Investigate problems before importing files.
102110     exit 1
102111 fi
```

102112 To verify that all files in the current directory hierarchy could be moved to any system
102113 conforming to the System Interfaces volume of POSIX.1-2017 that also supports the `pax` utility:

```
102114 find . -exec pathchk -p -P {} +
102115 if [ $? -eq 0 ]
102116 then
102117     pax -w -f ../archive .
102118 else
102119     echo Portable archive cannot be created.
102120     exit 1
102121 fi
```

102122 To verify that a user-supplied pathname names a readable file and that the application can create
102123 a file extending the given path without truncation and without overwriting any existing file:

```
102124 case $- in
102125     *C*)   reset="";;
102126     *)    reset="set +C"
102127           set -C;;
102128 esac
102129 test -r "$path" && pathchk "$path.out" &&
102130 rm "$path.out" > "$path.out"
102131 if [ $? -ne 0 ]; then
102132     printf "%s: %s not found or %s.out fails \
102133 creation checks.\n" $0 "$path" "$path"
102134     $reset      # Reset the noclobber option in case a trap
102135                # on EXIT depends on it.
102136     exit 1
102137 fi
102138 $reset
102139 PROCESSING < "$path" > "$path.out"
```

102140 The following assumptions are made in this example:

- 102141 1. **PROCESSING** represents the code that is used by the application to use `$path` once it is
102142 verified that `$path.out` works as intended.

- 102143 2. The state of the *noclobber* option is unknown when this code is invoked and should be set
 102144 on exit to the state it was in when this code was invoked. (The **reset** variable is used in
 102145 this example to restore the initial state.)
- 102146 3. Note the usage of:
- 102147 `rm "$path.out" > "$path.out"`
- 102148 a. The *pathchk* command has already verified, at this point, that **\$path.out** is not
 102149 truncated.
- 102150 b. With the *noclobber* option set, the shell verifies that **\$path.out** does not already exist
 102151 before invoking *rm*.
- 102152 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application
 102153 can create the file again in the **PROCESSING** step.
- 102154 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:
- 102155 `rm "$path.out" > "$path.out"`
 102156 should be replaced with:
 102157 `> "$path.out"`
- 102158 which verifies that the file did not already exist, but leaves **\$path.out** in place for
 102159 use by **PROCESSING**.

102160 RATIONALE

102161 The *pathchk* utility was new for the ISO POSIX-2:1993 standard. It, along with the *set*
 102162 `-C(noclobber)` option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that
 102163 appeared in early proposals. All of these utilities were attempts to solve several common
 102164 problems:

102165 Verify the validity (for several different definitions of “valid”) of a pathname supplied by a
 102166 user, generated by an application, or imported from an external source.

102167 Atomically create a file.

102168 Perform various string handling functions to generate a temporary filename.

102169 The *create* utility, included in an early proposal, provided checking and atomic creation in a
 102170 single invocation of the utility; these are orthogonal issues and need not be grouped into a single
 102171 utility. Note that the *noclobber* option also provides a way of creating a lock for process
 102172 synchronization; since it provides an atomic *create*, there is no race between a test for existence
 102173 and the following creation if it did not exist.

102174 Having a function like *tmpnam()* in the ISO C standard is important in many high-level
 102175 languages. The shell programming language, however, has built-in string manipulation
 102176 facilities, making it very easy to construct temporary filenames. The names needed obviously
 102177 depend on the application, but are frequently of a form similar to:

102178 `$TMPDIR/application_abbreviation$$.suffix`

102179 In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop
 102180 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as
 102181 applications trying to use the same filename name space are cooperating on the use of files after
 102182 they have been created.

102183 For historical purposes, `-p` does not check for the use of the <hyphen-minus> character as the
 102184 first character in a component of the pathname, or for an empty *pathname* operand.

102185 **FUTURE DIRECTIONS**

102186 None.

102187 **SEE ALSO**102188 [Section 2.7](#) (on page 2360), [set](#) (on page 2409), [test](#)102189 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216), [<limits.h>](#)102190 **CHANGE HISTORY**

102191 First released in Issue 4.

102192 **Issue 7**

102193 Austin Group Interpretations 1003.1-2001 #039, #040, and #094 are applied.

102194 SD5-XCU-ERN-121 is applied, updating the EXAMPLES section.

102195 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0127 [291] is applied.

102196 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0150 [584] and XCU/TC2-2008/0151
102197 [584] are applied.

102198 **NAME**

102199 pax — portable archive interchange

102200 **SYNOPSIS**102201 pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive] [-s replstr]...
102202 [pattern...]102203 pax -r[-c|-n] [-dikuv] [-H|-L] [-f archive] [-o options]... [-p string]...
102204 [-s replstr]... [pattern...]102205 pax -w [-dituvX] [-H|-L] [-b blocksize] [[-a] [-f archive]] [-o options]...
102206 [-s replstr]... [-x format] [file...]102207 pax -r -w [-diklntuvX] [-H|-L] [-o options]... [-p string]...
102208 [-s replstr]... [file...] directory102209 **DESCRIPTION**102210 The *pax* utility shall read, write, and write lists of the members of archive files and copy
102211 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.102212 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations
102213 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,
102214 corresponding respectively to the four forms shown in the SYNOPSIS section.102215 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of
102216 the members of the archive file read from the standard input, with pathnames
102217 matching the specified patterns, to standard output. If a named file is of type
102218 directory, the file hierarchy rooted at that file shall be listed as well.102219 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of
102220 the archive file read from the standard input, with pathnames matching the
102221 specified patterns. If an extracted file is of type directory, the file hierarchy rooted
102222 at that file shall be extracted as well. The extracted files shall be created performing
102223 pathname resolution with the directory in which *pax* was invoked as the current
102224 working directory.102225 If an attempt is made to extract a directory when the directory already exists, this
102226 shall not be considered an error. If an attempt is made to extract a FIFO when the
102227 FIFO already exists, this shall not be considered an error.102228 The ownership, access, and modification times, and file mode of the restored files
102229 are discussed under the *-p* option.102230 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of
102231 the *file* operands to the standard output in an archive format. If no *file* operands are
102232 specified, a list of files to copy, one per line, shall be read from the standard input
102233 and each entry in this list shall be processed as if it had been a *file* operand on the
102234 command line. A file of type directory shall include all of the files in the file
102235 hierarchy rooted at the file.102236 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands
102237 to the destination directory.102238 If no *file* operands are specified, a list of files to copy, one per line, shall be read
102239 from the standard input. A file of type directory shall include all of the files in the
102240 file hierarchy rooted at the file.102241 The effect of the **copy** shall be as if the copied files were written to a *pax* format
102242 archive file and then subsequently extracted, except that copying of sockets may be

102243 supported even if archiving them in write mode is not supported, and that there
 102244 may be hard links between the original and the copied files. If the destination
 102245 directory is a subdirectory of one of the files to be copied, the results are
 102246 unspecified. If the destination directory is a file of a type not defined by the System
 102247 Interfaces volume of POSIX.1-2017, the results are implementation-defined;
 102248 otherwise, it shall be an error for the file named by the *directory* operand not to
 102249 exist, not be writable by the user, or not be a file of type directory.

102250 In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member,
 102251 *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces
 102252 volume of POSIX.1-2017, called with the following arguments:

102253 The intermediate directory used as the *path* argument

102254 The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode*
 102255 argument

102256 If any specified *pattern* or *file* operands are not matched by at least one file or archive member,
 102257 *pax* shall write a diagnostic message to standard error for each one that did not match and exit
 102258 with a non-zero exit status.

102259 The archive formats described in the EXTENDED DESCRIPTION section shall be automatically
 102260 detected on input. The default output archive format shall be implementation-defined.

102261 A single archive can span multiple files. The *pax* utility shall determine, in an implementation-
 102262 defined manner, what file to read or write as the next file.

102263 If the selected archive format supports the specification of linked files, it shall be an error if these
 102264 files cannot be linked when the archive is extracted. For archive formats that do not store file
 102265 contents with each name that causes a hard link, if the file that contains the data is not extracted
 102266 during this *pax* session, either the data shall be restored from the original file, or a diagnostic
 102267 message shall be displayed with the name of a file that can be used to extract the data. In
 102268 traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited
 102269 directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall
 102270 write a diagnostic message to standard error and shall terminate.

102271 OPTIONS

102272 The *pax* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the order of
 102273 presentation of the **-o**, **-p**, and **-s** options is significant.

102274 The following options shall be supported:

102275 **-r** Read an archive file from standard input.

102276 **-w** Write files to the standard output in the specified archive format.

102277 **-a** Append files to the end of the archive. It is implementation-defined which devices
 102278 on the system support appending. Additional file formats unspecified by this
 102279 volume of POSIX.1-2017 may impose restrictions on appending.

102280 **-b** *blocksize* Block the output at a positive decimal integer number of bytes per write to the
 102281 archive file. Devices and archive formats may impose restrictions on blocking.
 102282 Blocking shall be automatically determined on input. Conforming applications
 102283 shall not specify a *blocksize* value larger than 32256. Default blocking when
 102284 creating archives depends on the archive format. (See the **-x** option below.)

102285 **-c** Match all file or archive members except those specified by the *pattern* or *file*
 102286 operands.

102287	-d	Cause files of type directory being copied or archived or archive members of type directory being extracted or listed to match only the file or archive member itself and not the file hierarchy rooted at the file.
102288		
102289		
102290	-f <i>archive</i>	Specify the pathname of the input or output archive, overriding the default standard input (in list or read modes) or standard output (write mode).
102291		
102292	-H	If a symbolic link referencing a file of type directory is specified on the command line, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line, then <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.
102293		
102294		
102295		
102296		
102297		
102298		
102299	-i	Interactively rename files or archive members. For each archive member matching a <i>pattern</i> operand or file matching a <i>file</i> operand, a prompt shall be written to the file /dev/tty . The prompt shall contain the name of the file or archive member, but the format is otherwise unspecified. A line shall then be read from /dev/tty . If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The <i>pax</i> utility shall immediately exit with a non-zero exit status if end-of-file is encountered when reading a response or if /dev/tty cannot be opened for reading and writing.
102300		
102301		
102302		
102303		
102304		
102305		
102306		
102307		
102308		
102309		The results of extracting a hard link to a file that has been renamed during extraction are unspecified.
102310		
102311	-k	Prevent the overwriting of existing files.
102312	-l	(The letter ell.) In copy mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with -H or -L, when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither -H nor -L is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.
102313		
102314		
102315		
102316		
102317		
102318		
102319	-L	If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which <i>pax</i> can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, <i>pax</i> shall archive the file referenced by the link, using the name of the link. The default behavior, when neither -H or -L are specified, shall be to archive the symbolic link itself.
102320		
102321		
102322		
102323		
102324		
102325		
102326		
102327	-n	Select the first archive member that matches each <i>pattern</i> operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).
102328		
102329		
102330	-o <i>options</i>	Provide information to the implementation to modify the algorithm for extracting or writing files. The value of <i>options</i> shall consist of one or more <comma>-separated keywords of the form:
102331		
102332		
102333		

```
keyword[[:]=value] [,keyword[[:]=value], ...]
```

102334 Some keywords apply only to certain file formats, as indicated with each
 102335 description. Use of keywords that are inapplicable to the file format being
 102336 processed produces undefined results.

102337 Keywords in the *options* argument shall be a string that would be a valid portable
 102338 filename as described in XBD [Section 3.282](#) (on page 79).

102339 **Note:** Keywords are not expected to be filenames, merely to follow the same character
 102340 composition rules as portable filenames.

102341 Keywords can be preceded with white space. The *value* field shall consist of zero or
 102342 more characters; within *value*, the application shall precede any literal <comma>
 102343 with a <backslash>, which shall be ignored, but preserves the <comma> as part of
 102344 *value*. A <comma> as the final character, or a <comma> followed solely by white
 102345 space as the final characters, in *options* shall be ignored. Multiple **-o** options can be
 102346 specified; if keywords given to these multiple **-o** options conflict, the keywords
 102347 and values appearing later in command line sequence shall take precedence and
 102348 the earlier shall be silently ignored. The following keyword values of *options* shall
 102349 be supported for the file formats as indicated:

102350 **delete=pattern**

102351 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*
 102352 shall omit from extended header records that it produces any keywords
 102353 matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore
 102354 any keywords matching the string pattern in the extended header records. In
 102355 both cases, matching shall be performed using the pattern matching notation
 102356 described in [Section 2.13.1](#) (on page 2382) and [Section 2.13.2](#) (on page 2383).
 102357 For example:

102358 **-o delete=security.***

102359 would suppress security-related information. See [pax Extended Header](#) (on
 102360 page 3082) for extended header record keyword usage.

102361 When multiple **-odelete=pattern** options are specified, the patterns shall be
 102362 additive; all keywords matching the specified string patterns shall be omitted
 102363 from extended header records that *pax* produces.

102364 **exthdr.name=string**

102365 (Applicable only to the **-x pax** format.) This keyword allows user control over
 102366 the name that is written into the **ustar** header blocks for the extended header
 102367 produced under the circumstances described in [pax Header Block](#) (on page
 102368 3081). The name shall be the contents of *string*, after the following character
 102369 substitutions have been made:

<i>string</i> Includes:	Replaced by:
%d	The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname.
%f	The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.
%p	The process ID of the <i>pax</i> process.
%%	A '%' character.

102378 Any other '%' characters in *string* produce undefined results.

102379 If no **-o exthdr.name=string** is specified, *pax* shall use the following default

102380 value:
 102381 %d/PaxHeaders.%p/%f

102382 **globexthdr.name=string**

102383 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode
 102384 with the appropriate options, *pax* shall create global extended header records
 102385 with **ustar** header blocks that will be treated as regular files by previous
 102386 versions of *pax*. This keyword allows user control over the name that is
 102387 written into the **ustar** header blocks for global extended header records. The
 102388 name shall be the contents of *string*, after the following character substitutions
 102389 have been made:

<i>string</i> Includes:	Replaced by:
%n	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
%p	The process ID of the <i>pax</i> process.
%%	A '%' character.

102396 Any other '%' characters in *string* produce undefined results.

102397 If no **-o globexthdr.name=string** is specified, *pax* shall use the following
 102398 default value:

102399 \$TMPDIR/GlobalHead.%p.%n

102400 where *\$TMPDIR* represents the value of the *TMPDIR* environment variable. If
 102401 *TMPDIR* is not set, *pax* shall use **/tmp**.

102402 **invalid=action**

102403 (Applicable only to the **-x pax** format.) This keyword allows user control over
 102404 the action *pax* takes upon encountering values in an extended header record
 102405 that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list**
 102406 mode, cannot be written in the codeset and current locale of the
 102407 implementation. The following are invalid values that shall be recognized by
 102408 *pax*:

102409 **fn'lead** or **copy** mode, a filename or link name that contains character
 102410 encodings invalid in the destination hierarchy. (For example, the name
 102411 may contain embedded NULs.)

102412 **fn'lead** or **copy** mode, a filename or link name that is longer than the
 102413 maximum allowed in the destination hierarchy (for either a pathname
 102414 component or the entire pathname).

102415 **fn'list** mode, any character string value (filename, link name, user name,
 102416 and so on) that cannot be written in the codeset and current locale of the
 102417 implementation.

102418 The following mutually-exclusive values of the *action* argument are supported:

102419 **binary** In **write** mode, *pax* shall generate a **hdrcharset=BINARY**
 102420 extended header record for each file with a filename, link name,
 102421 group name, owner name, or any other field in an extended
 102422 header record that cannot be translated to the UTF-8 codeset,
 102423 allowing the archive to contain the files with unencoded
 102424 extended header record values. In **read** or **copy** mode, *pax* shall

102425 use the values specified in the header without translation,
 102426 regardless of whether this may overwrite an existing file with a
 102427 valid name. In **list** mode, *pax* shall behave identically to the
 102428 **bypass** action.

102429 **bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no
 102430 change to the destination hierarchy. In **list** mode, *pax* shall write
 102431 all requested valid values for the file, but its method for writing
 102432 invalid values is unspecified.

102433 **rename** In **read** or **copy** mode, *pax* shall act as if the **-i** option were in
 102434 effect for each file with invalid filename or link name values,
 102435 allowing the user to provide a replacement name interactively.
 102436 In **list** mode, *pax* shall behave identically to the **bypass** action.

102437 **UTF-8** When used in **read**, **copy**, or **list** mode and a filename, link
 102438 name, owner name, or any other field in an extended header
 102439 record cannot be translated from the **pax** UTF-8 codeset format
 102440 to the codeset and current locale of the implementation, *pax* shall
 102441 use the actual UTF-8 encoding for the name. If a **hdrcharset**
 102442 extended header record is in effect for this file, the character set
 102443 specified by that record shall be used instead of UTF-8. If a
 102444 **hdrcharset=BINARY** extended header record is in effect for this
 102445 file, no translation shall be performed.

102446 **write** In **read** or **copy** mode, *pax* shall write the file, translating the
 102447 name, regardless of whether this may overwrite an existing file
 102448 with a valid name. In **list** mode, *pax* shall behave identically to
 102449 the **bypass** action.

102450 If no **-o invalid=option** is specified, *pax* shall act as if **-o invalid=bypass** were
 102451 specified. Any overwriting of existing files that may be allowed by the
 102452 **-o invalid=** actions shall be subject to permission (**-p**) and modification time
 102453 (**-u**) restrictions, and shall be suppressed if the **-k** option is also specified.

102454 **linkdata**
 102455 (Applicable only to the **-x pax** format.) In **write** mode, *pax* shall write the
 102456 contents of a file to the archive even when that file is merely a hard link to a
 102457 file whose contents have already been written to the archive.

102458 **listopt=format**
 102459 This keyword specifies the output format of the table of contents produced
 102460 when the **-v** option is specified in **list** mode. See [List Mode Format Specifications](#)
 102461 (on page 3076). To avoid ambiguity, the **listopt=format** shall be
 102462 the only or final **keyword=value** pair in a **-o** option-argument; all characters
 102463 in the remainder of the option-argument shall be considered part of the format
 102464 string. When multiple **-olistopt=format** options are specified, the format
 102465 strings shall be considered a single, concatenated string, evaluated in
 102466 command line order.

102467 **times**
 102468 (Applicable only to the **-x pax** format.) When used in **write** or **copy** mode, *pax*
 102469 shall include **atime** and **mtime** extended header records for each file. See [pax Extended Header File Times](#)
 102470 (on page 3085).

102471 In addition to these keywords, if the **-x pax** format is specified, any of the

102472 keywords and values defined in [pax Extended Header](#) (on page 3082), including
 102473 implementation extensions, can be used in **-o** option-arguments, in either of two
 102474 modes:

102475 **keyword=***value*

102476 When used in **write** or **copy** mode, these keyword/value pairs shall be
 102477 included at the beginning of the archive as **typeflag g** global extended header
 102478 records. When used in **read** or **list** mode, these keyword/value pairs shall act
 102479 as if they had been at the beginning of the archive as **typeflag g** global
 102480 extended header records.

102481 **keyword:=***value*

102482 When used in **write** or **copy** mode, these keyword/value pairs shall be
 102483 included as records at the beginning of a **typeflag x** extended header for each
 102484 file. (This shall be equivalent to the <equals-sign> form except that it creates
 102485 no **typeflag g** global extended header records.) When used in **read** or **list**
 102486 mode, these keyword/value pairs shall act as if they were included as records
 102487 at the end of each extended header; thus, they shall override any global or file-
 102488 specific extended header record keywords of the same names. For example, in
 102489 the command:

```
102490 pax -r -o "  
102491 gname:=mygroup,  
102492 " <archive
```

102493 the group name will be forced to a new value for all files read from the
 102494 archive.

102495 The precedence of **-o** keywords over various fields in the archive is described in
 102496 [pax Extended Header Keyword Precedence](#) (on page 3085). If the **-o**
 102497 **delete=***pattern*, **-o keyword=***value*, or **-o keyword:=***value* options are used to
 102498 override or remove any extended header data needed to find files in an archive
 102499 (e.g., **-o delete=size** for a file whose size cannot be represented in a **ustar**
 102500 header or **-o size=100** for a file whose size is not 100 bytes), the behavior is
 102501 undefined.

102502 **-p** *string* Specify one or more file characteristic options (privileges). The *string* option-
 102503 argument shall be a string specifying file characteristics to be retained or discarded
 102504 on extraction. The string shall consist of the specification characters *a*, *e*, *m*, *o*, and
 102505 *p*. Other implementation-defined characters can be included. Multiple
 102506 characteristics can be concatenated within the same string and multiple **-p** options
 102507 can be specified. The meaning of the specification characters are as follows:

- 102508 a Do not preserve file access times.
- 102509 e Preserve the user ID, group ID, file mode bits (see XBD [Section 3.169](#), on page
102510 60), access time, modification time, and any other implementation-defined file
102511 characteristics.
- 102512 m Do not preserve file modification times.
- 102513 o Preserve the user ID and group ID.
- 102514 p Preserve the file mode bits. Other implementation-defined file mode attributes
102515 may be preserved.

102516 In the preceding list, “preserve” indicates that an attribute stored in the archive
 102517 shall be given to the extracted file, subject to the permissions of the invoking

102518 process. The access and modification times of the file shall be preserved unless
 102519 otherwise specified with the `-p` option or not stored in the archive. All attributes
 102520 that are not preserved shall be determined as part of the normal file creation action
 102521 (see [Section 1.1.1.4](#), on page 2328).

102522 If neither the `e` nor the `o` specification character is specified, or the user ID and
 102523 group ID are not preserved for any reason, *pax* shall not set the `S_ISUID` and
 102524 `S_ISGID` bits of the file mode.

102525 If the preservation of any of these items fails for any reason, *pax* shall write a
 102526 diagnostic message to standard error. Failure to preserve these items shall affect
 102527 the final exit status, but shall not cause the extracted file to be deleted.

102528 If file characteristic letters in any of the *string* option-arguments are duplicated or
 102529 conflict with each other, the ones given last shall take precedence. For example, if
 102530 `-p eme` is specified, file modification times are preserved.

102531 `-s replstr` Modify file or archive member names named by *pattern* or *file* operands according
 102532 to the substitution expression *replstr*, using the syntax of the *ed* utility. The concepts
 102533 of “address” and “line” are meaningless in the context of the *pax* utility, and shall
 102534 not be supplied. The format shall be:

102535 `-s /old/new/[gp]`

102536 where as in *ed*, *old* is a basic regular expression and *new* can contain an
 102537 `<ampersand>`, `'\n'` (where *n* is a digit) back-references, or subexpression
 102538 matching. The *old* string shall also be permitted to contain `<newline>` characters.

102539 Any non-null character can be used as a delimiter (`'/'` shown here). Multiple `-s`
 102540 expressions can be specified; the expressions shall be applied in the order
 102541 specified, terminating with the first successful substitution. The optional trailing
 102542 `'g'` is as defined in the *ed* utility. The optional trailing `'p'` shall cause successful
 102543 substitutions to be written to standard error. File or archive member names that
 102544 substitute to the empty string shall be ignored when reading and writing archives.

102545 `-t` When reading files from the file system, and if the user has the permissions
 102546 required by *utime()* to do so, set the access time of each file read to the access time
 102547 that it had before being read by *pax*.

102548 `-u` Ignore files that are older (having a less recent file modification time) than a pre-
 102549 existing file or archive member with the same name. In **read** mode, an archive
 102550 member with the same name as a file in the file system shall be extracted if the
 102551 archive member is newer than the file. In **write** mode, an archive file member with
 102552 the same name as a file in the file system shall be superseded if the file is newer
 102553 than the archive member. If `-a` is also specified, this is accomplished by appending
 102554 to the archive; otherwise, it is unspecified whether this is accomplished by actual
 102555 replacement in the archive or by appending to the archive. In **copy** mode, the file
 102556 in the destination hierarchy shall be replaced by the file in the source hierarchy or
 102557 by a link to the file in the source hierarchy if the file in the source hierarchy is
 102558 newer.

102559 `-v` In **list** mode, produce a verbose table of contents (see the **STDOUT** section).
 102560 Otherwise, write archive member pathnames to standard error (see the **STDERR**
 102561 section).

- 102562 **-x format** Specify the output archive format. The *pax* utility shall support the following
102563 formats:
- 102564 **cpio** The **cpio** interchange format; see the EXTENDED DESCRIPTION
102565 section. The default *blocksize* for this format for character special
102566 archive files shall be 5120. Implementations shall support all
102567 *blocksize* values less than or equal to 32256 that are multiples of 512.
- 102568 **pax** The **pax** interchange format; see the EXTENDED DESCRIPTION
102569 section. The default *blocksize* for this format for character special
102570 archive files shall be 5120. Implementations shall support all
102571 *blocksize* values less than or equal to 32256 that are multiples of 512.
- 102572 **ustar** The **tar** interchange format; see the EXTENDED DESCRIPTION
102573 section. The default *blocksize* for this format for character special
102574 archive files shall be 10240. Implementations shall support all
102575 *blocksize* values less than or equal to 32256 that are multiples of 512.
- 102576 Implementation-defined formats shall specify a default block size as well as any
102577 other block sizes supported for character special archive files.
- 102578 Any attempt to append to an archive file in a format different from the existing
102579 archive format shall cause *pax* to exit immediately with a non-zero exit status.
- 102580 **-X** When traversing the file hierarchy specified by a pathname, *pax* shall not descend
102581 into directories that have a different device ID (*st_dev*; see the System Interfaces
102582 volume of POSIX.1-2017, *stat()*).
- 102583 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
102584 an error and the last option specified shall determine the behavior of the utility.
- 102585 The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)
102586 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-
102587 specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i**
102588 options shall modify, in that order, the names of the selected files. The **-v** option shall write
102589 names resulting from these modifications.
- 102590 In **write** mode, the files shall be selected based on the user-specified pathnames as modified by
102591 the **-n** and **-u** options. Then, any **-s** and **-i** options shall modify, in that order, the names of
102592 these selected files. The **-v** option shall write names resulting from these modifications.
- 102593 If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is
102594 newer than the file to which it is compared.
- 102595 **List Mode Format Specifications**
- 102596 In **list** mode with the **-o listopt=format** option, the *format* argument shall be applied for each
102597 selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file.
102598 The *format* argument shall be used as the *format* string described in XBD Chapter 5 (on page 121),
102599 with the exceptions 1. through 6. defined in the EXTENDED DESCRIPTION section of *printf*,
102600 plus the following exceptions:
- 102601 7. The sequence (*keyword*) can occur before a format conversion specifier. The conversion
102602 argument is defined by the value of *keyword*. The implementation shall support the
102603 following keywords:

102604 ‡ *ny*Of the Field Name entries in [Table 4-14](#) (on page 3086) and [Table 4-16](#) (on page
102605 3090). The implementation may support the *cpio* keywords without the leading *c_* in
102606 addition to the form required by [Table 4-16](#) (on page 3090).

102607 ‡ *ny*A keyword defined for the extended header in [pax Extended Header](#) (on page
102608 3082).

102609 ‡ *ny*A keyword provided as an implementation-defined extension within the extended
102610 header defined in [pax Extended Header](#) (on page 3082).

102611 For example, the sequence "*%(charset)s*" is the string value of the name of the character
102612 set in the extended header.

102613 The result of the keyword conversion argument shall be the value from the applicable
102614 header field or extended header, without any trailing NULs.

102615 All keyword values used as conversion arguments shall be translated from the UTF-8
102616 encoding (or alternative encoding specified by any **hdrcharset** extended header record) to
102617 the character set appropriate for the local file system, user database, and so on, as
102618 applicable.

102619 8. An additional conversion specifier character, **T**, shall be used to specify time formats. The **T**
102620 conversion specifier character can be preceded by the sequence (*keyword=subformat*), where
102621 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**
102622 and the default subformat shall be:

102623 %b %e %H:%M %Y

102624 9. An additional conversion specifier character, **M**, shall be used to specify the file mode string
102625 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.
102626 For example, *% . 1M* writes the single character corresponding to the *<entry type>* field of the
102627 *ls -l* command.

102628 10. An additional conversion specifier character, **D**, shall be used to specify the device for block
102629 or special files, if applicable, in an implementation-defined format. If not applicable, and
102630 (*keyword*) is specified, then this conversion shall be equivalent to *%(keyword)u*. If not
102631 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to *<space>*.

102632 11. An additional conversion specifier character, **F**, shall be used to specify a pathname. The **F**
102633 conversion character can be preceded by a sequence of *<comma>*-separated keywords:

102634 (*keyword*[,*keyword*] . . .)

102635 The values for all the keywords that are non-null shall be concatenated together, each
102636 separated by a *'/'*. The default shall be (**path**) if the keyword **path** is defined; otherwise,
102637 the default shall be (**prefix,name**).

102638 12. An additional conversion specifier character, **L**, shall be used to specify a symbolic link
102639 expansion. If the current file is a symbolic link, then *%L* shall expand to:

102640 "%s -> %s", *<value of keyword>*, *<contents of link>*

102641 Otherwise, the *%L* conversion specification shall be the equivalent of *%F*.

102642 OPERANDS

102643 The following operands shall be supported:

102644 *directory* The destination directory pathname for **copy** mode.

102645	<i>file</i>	A pathname of a file to be copied or archived.
102646	<i>pattern</i>	A pattern matching one or more pathnames of archive members. A pattern must be given in the name-generating notation of the pattern matching notation in Section 2.13 (on page 2382), including the filename expansion rules in Section 2.13.3 (on page 2383). The default, if no <i>pattern</i> is specified, is to select all members in the archive.
102647		
102648		
102649		
102650		
102651	STDIN	
102652		In write mode, the standard input shall be used only if no <i>file</i> operands are specified. It shall be a file containing a list of pathnames, each terminated by a <newline> character.
102653		
102654		In list and read modes, if -f is not specified, the standard input shall be an archive file.
102655		Otherwise, the standard input shall not be used.
102656	INPUT FILES	
102657		The input file named by the <i>archive</i> option-argument, or standard input when the archive is read from there, shall be a file formatted according to one of the specifications in the EXTENDED DESCRIPTION section or some other implementation-defined format.
102658		
102659		
102660		The file <code>/dev/tty</code> shall be used to write prompts and read responses.
102661	ENVIRONMENT VARIABLES	
102662		The following environment variables shall affect the execution of <i>pax</i> :
102663	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 174) the precedence of internationalization variables used to determine the values of locale categories.)
102664		
102665		
102666	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
102667		
102668	<i>LC_COLLATE</i>	
102669		Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the pattern matching expressions for the <i>pattern</i> operand, the basic regular expression for the -s option, and the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category.
102670		
102671		
102672		
102673		
102674	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes used in the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category, and pattern matching.
102675		
102676		
102677		
102678		
102679	<i>LC_MESSAGES</i>	
102680		Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of diagnostic messages and prompts written to standard error.
102681		
102682		
102683	<i>LC_TIME</i>	Determine the format and contents of date and time strings when the -v option is specified.
102684		
102685	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
102686	<i>TMPDIR</i>	Determine the pathname that provides part of the default global extended header record file, as described for the -o globexthdr= keyword in the OPTIONS section.
102687		

102688 *TZ* Determine the timezone used to calculate date and time strings when the `-v` option
 102689 is specified. If *TZ* is unset or null, an unspecified default timezone shall be used.

102690 ASYNCHRONOUS EVENTS

102691 Default.

102692 STDOUT

102693 In **write** mode, if `-f` is not specified, the standard output shall be the archive formatted
 102694 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other
 102695 implementation-defined format (see `-x format`).

102696 In **list** mode, when the `-olistopt=format` has been specified, the selected archive members shall
 102697 be written to standard output using the format described under [List Mode Format Specifications](#)
 102698 (on page 3076). In **list** mode without the `-olistopt=format` option, the table of contents of the
 102699 selected archive members shall be written to standard output using the following format:

102700 "%s\n", <pathname>

102701 If the `-v` option is specified in **list** mode, the table of contents of the selected archive members
 102702 shall be written to standard output using the following formats.

102703 For pathnames representing hard links to previous members of the archive:

102704 "%sΔ==Δ%s\n", <ls -l listing>, <linkname>

102705 For all other pathnames:

102706 "%s\n", <ls -l listing>

102707 where <ls -l listing> shall be the format specified by the *ls* utility with the `-l` option. When
 102708 writing pathnames in this format, it is unspecified what is written for fields for which the
 102709 underlying archive format does not have the correct information, although the correct number of
 102710 <blank>-separated fields shall be written.

102711 In **list** mode, standard output shall not be buffered more than a pathname (plus any associated
 102712 information and a <newline> terminator) at a time.

102713 STDERR

102714 If `-v` is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the
 102715 standard error output using the following format:

102716 "%s\n", <pathname>

102717 These pathnames shall be written as soon as processing is begun on the file or archive member,
 102718 and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, is
 102719 written when the file has been read or written.

102720 If the `-s` option is specified, and the replacement string has a trailing 'p', substitutions shall be
 102721 written to standard error in the following format:

102722 "%sΔ>>Δ%s\n", <original pathname>, <new pathname>

102723 In all operating modes of *pax*, optional messages of unspecified format concerning the input
 102724 archive format and volume number, the number of files, blocks, volumes, and media parts as
 102725 well as other diagnostic messages may be written to standard error.

102726 In all formats, for both standard output and standard error, it is unspecified how non-printable
 102727 characters in pathnames or link names are written.

102728 When using the `-xpax` archive format, if a filename, link name, group name, owner name, or any
 102729 other field in an extended header record cannot be translated between the codeset in use for that

102730 extended header record and the character set of the current locale, *pax* shall write a diagnostic
102731 message to standard error, shall process the file as described for the **-o invalid=** option, and then
102732 shall continue processing with the next file.

102733 OUTPUT FILES

102734 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the
102735 copied output files shall be the type of the file being copied. In either mode, existing files in the
102736 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),
102737 and invalid-value (**-o invalid=**) tests allow it.

102738 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted
102739 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other
102740 implementation-defined format.

102741 EXTENDED DESCRIPTION

102742 **pax Interchange Format**

102743 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The
102744 physical layout of the archive shall be identical to the **ustar** format described in [ustar](#)
102745 [Interchange Format](#) (on page 3086). Each file archived shall be represented by the following
102746 sequence:

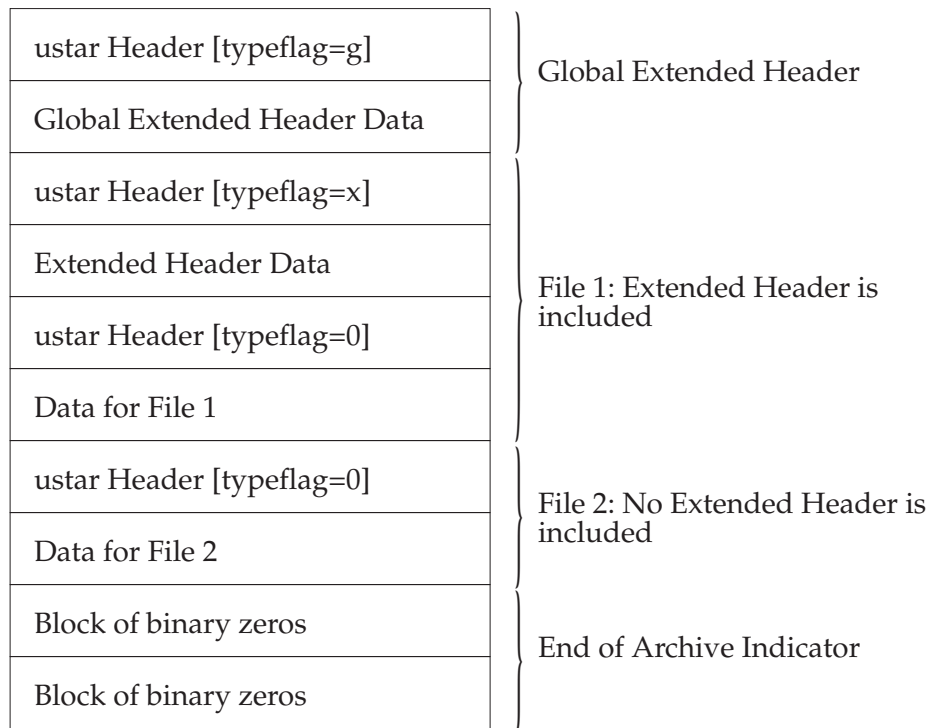
102747 An optional header block with extended header records. This header block is of the form
102748 described in [pax Header Block](#) (on page 3081), with a *typeflag* value of **x** or **g**. The
102749 extended header records, described in [pax Extended Header](#) (on page 3082), shall be
102750 included as the data for this header block.

102751 A header block that describes the file. Any fields in the preceding optional extended
102752 header shall override the associated fields in this header block for this file.

102753 Zero or more blocks that contain the contents of the file.

102754 At the end of the archive file there shall be two 512-byte blocks filled with binary zeros,
102755 interpreted as an end-of-archive indicator.

102756 A schematic of an example archive with global extended header records and two actual files is
102757 shown in [Figure 4-1](#) (on page 3081). In the example, the second file in the archive has no
102758 extended header preceding it, presumably because it has no need for extended attributes.



102759

Figure 4-1 pax Format Archive Example

102760

pax Header Block

102761

The **pax** header block shall be identical to the **ustar** header block described in [ustar Interchange Format](#) (on page 3086), except that two additional *typeflag* values are defined:

102762

102763

× Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in [pax Extended Header](#) (on page 3082).

102764

102765

102766

g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in [pax Extended Header](#) (on page 3082). Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag* g global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

102767

102768

102769

102770

102771

102772

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2:1993 standard, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

102773

102774

102775

102776

102777

102778

A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit one) (hard link) may be included, which means that the size field may be greater than zero. Archives created by *pax -o linkdata* shall include these data blocks with the hard links.

102779

102780

102781 **pax Extended Header**

102782 A **pax** extended header contains values that are inappropriate for the **ustar** header block because
 102783 of limitations in that format: fields requiring a character encoding other than that described in
 102784 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**
 102785 header, and fields whose format or length do not fit the requirements of the **ustar** header. The
 102786 values in an extended header add attributes to the following file (or files; see the description of
 102787 the *typeflag* **g** header block) or override values in the following header block(s), as indicated in
 102788 the following list of keywords.

102789 An extended header shall consist of one or more records, each constructed as follows:

102790 "%d %s=%s\n", <length>, <keyword>, <value>

102791 The extended header records shall be encoded according to the ISO/IEC 10646-1:2000 standard
 102792 UTF-8 encoding. The <length> field, <blank>, <equals-sign>, and <newline> shown shall be
 102793 limited to the portable character set, as encoded in UTF-8. The <keyword> fields can be any
 102794 UTF-8 characters. The <length> field shall be the decimal length of the extended header record
 102795 in octets, including the trailing <newline>. If there is a **hdrcharset** extended header in effect for
 102796 a file, the *value* field for any **gname**, **linkpath**, **path**, and **uname** extended header records shall be
 102797 encoded using the character set specified by the **hdrcharset** extended header record; otherwise,
 102798 the *value* field shall be encoded using UTF-8. The *value* field for all other keywords specified by
 102799 POSIX.1-2017 shall be encoded using UTF-8.

102800 The <keyword> field shall be one of the entries from the following list or a keyword provided as
 102801 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and
 102802 periods are reserved for future standardization. A keyword shall not include an <equals-sign>.
 102803 (In the following list, the notations "file(s)" or "block(s)" is used to acknowledge that a keyword
 102804 affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after
 102805 *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode
 102806 shall apply only when such a record has not already been provided through the use of the **-o**
 102807 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with
 102808 applicable extended header records and then extracted.)

102809 **atime** The file access time for the following file(s), equivalent to the value of the *st_atime*
 102810 member of the **stat** structure for a file, as described by the *stat()* function. The
 102811 access time shall be restored if the process has appropriate privileges required to
 102812 do so. The format of the <value> shall be as described in [pax Extended Header File](#)
 102813 [Times](#) (on page 3085).

102814 **charset** The name of the character set used to encode the data in the following file(s). The
 102815 entries in the following table are defined to refer to known standards; additional
 102816 names may be agreed on between the originator and recipient.

102817
102818
102819
102820
102821
102822
102823
102824
102825
102826
102827
102828
102829
102830
102831
102832
102833
102834

<value>	Formal Standard
ISO-IRΔ646Δ1990	ISO/IEC 646:1990
ISO-IRΔ8859Δ1Δ1998	ISO/IEC 8859-1:1998
ISO-IRΔ8859Δ2Δ1999	ISO/IEC 8859-2:1999
ISO-IRΔ8859Δ3Δ1999	ISO/IEC 8859-3:1999
ISO-IRΔ8859Δ4Δ1998	ISO/IEC 8859-4:1998
ISO-IRΔ8859Δ5Δ1999	ISO/IEC 8859-5:1999
ISO-IRΔ8859Δ6Δ1999	ISO/IEC 8859-6:1999
ISO-IRΔ8859Δ7Δ1987	ISO/IEC 8859-7:1987
ISO-IRΔ8859Δ8Δ1999	ISO/IEC 8859-8:1999
ISO-IRΔ8859Δ9Δ1999	ISO/IEC 8859-9:1999
ISO-IRΔ8859Δ10Δ1998	ISO/IEC 8859-10:1998
ISO-IRΔ8859Δ13Δ1998	ISO/IEC 8859-13:1998
ISO-IRΔ8859Δ14Δ1998	ISO/IEC 8859-14:1998
ISO-IRΔ8859Δ15Δ1999	ISO/IEC 8859-15:1999
ISO-IRΔ10646Δ2000	ISO/IEC 10646:2000
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

102835
102836
102837

The encoding is included in an extended header for information only; when *pax* is used as described in POSIX.1-2017, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

102838
102839

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

102840
102841

comment A series of characters used as a comment. All characters in the <value> field shall be ignored by *pax*.

102842
102843
102844
102845
102846

gid The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

102847
102848
102849
102850
102851
102852
102853
102854
102855
102856

gname The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the characters cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the **-oinvalid=binary** option is specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

102857
102858
102859
102860

hdrcharset The name of the character set used to encode the value field of the **gname**, **linkpath**, **path**, and **uname** *pax* extended header records. The entries in the following table are defined to refer to known standards; additional names may be agreed between the originator and the recipient.

102861
102862
102863

<value>	Formal Standard
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

102864
102865
102866

If no **hdrcharset** extended header record is specified, the default character set used to encode all values in extended header records shall be the ISO/IEC 10646-1:2000 standard UTF-8 encoding.

102867
102868

The **BINARY** entry indicates that all values recorded in extended headers for affected files are unencoded binary data from the underlying system.

102869
102870
102871
102872
102873
102874
102875
102876
102877
102878
102879

linkpath

The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created. If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it shall be a symbolic link and the **linkpath** value shall be the contents of the symbolic link. The *pax* utility shall translate the name of the link (contents of the symbolic link) from the encoding in the header to the character set appropriate for the local file system. When used in **write** or **copy** mode, *pax* shall include a **linkpath** extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

102880
102881
102882
102883
102884
102885

mtime

The file modification time of the following file(s), equivalent to the value of the *st_mtime* member of the **stat** structure for a file, as described in the *stat()* function. This record shall override the *mtime* field in the following header block(s). The modification time shall be restored if the process has appropriate privileges required to do so. The format of the <value> shall be as described in [pax Extended Header File Times](#) (on page 3085).

102886
102887
102888
102889

path

The pathname of the following file(s). This record shall override the *name* and *prefix* fields in the following header block(s). The *pax* utility shall translate the pathname of the file from the encoding in the header to the character set appropriate for the local file system.

102890
102891
102892

When used in **write** or **copy** mode, *pax* shall include a *path* extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

102893

realtime.any The keywords prefixed by ``realtime.'' are reserved for future standardization.

102894

security.any The keywords prefixed by ``security.'' are reserved for future standardization.

102895
102896
102897
102898
102899

size

The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *size* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *size* extended header record for each file with a size value greater than 8 589 934 591 (octal 77 777 777 777).

102900
102901
102902
102903
102904

uid

The user ID of the file owner, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *uid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *uid* extended header record for each file whose owner ID is greater than 2 097 151 (octal 7777 777).

102905 **uname** The owner of the following file(s), formatted as a user name in the user database.
 102906 This record shall override the *uid* and *uname* fields in the following header block(s),
 102907 and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax*
 102908 shall translate the name from the encoding in the header record to the character set
 102909 appropriate for the user database on the receiving system. If any of the characters
 102910 cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the
 102911 **-oinvalid=binary** option is specified, the results are implementation-defined.
 102912 When used in **write** or **copy** mode, *pax* shall include a **uname** extended header
 102913 record for each file whose user name cannot be represented entirely with the letters
 102914 and digits of the portable character set.

102915 If the *<value>* field is zero length, it shall delete any header block field, previously entered
 102916 extended header value, or global extended header value of the same name.

102917 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a
 102918 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block
 102919 field.

102920 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the
 102921 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**
 102922 header block fields in Table 4-14 (on page 3086) shall apply to the extended header records.

102923 **pax Extended Header Keyword Precedence**

102924 This section describes the precedence in which the various header records and fields and
 102925 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or
 102926 **list** modes, it shall determine a file attribute in the following sequence:

- 102927 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step
 102928 7., if applicable, or ignored otherwise.
- 102929 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
- 102930 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
- 102931 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the
 102932 *<value>*. When extended header records conflict, the last one given in the header shall
 102933 take precedence.
- 102934 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
- 102935 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be
 102936 assigned the *<value>*. When global extended header records conflict, the last one given in
 102937 the global header shall take precedence.
- 102938 7. Otherwise, the attribute shall be determined from the **ustar** header block.

102939 **pax Extended Header File Times**

102940 The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's
 102941 modification time cannot be represented exactly in the **ustar** header logical record described in
 102942 **ustar Interchange Format** (on page 3086). This can occur if the time is out of **ustar** range, or if
 102943 the file system of the underlying implementation supports non-integer time granularities and
 102944 the time is not an integer. All of these time records shall be formatted as a decimal representation
 102945 of the time in seconds since the Epoch. If a *<period>* (' . ') decimal point character is present,
 102946 the digits to the right of the point shall represent the units of a subsecond timing granularity,
 102947 where the first digit is tenths of a second and each subsequent digit is a tenth of the previous
 102948 digit. In **read** or **copy** mode, the *pax* utility shall truncate the time of a file to the greatest value

102949 that is not greater than the input header file time. In **write** or **copy** mode, the *pax* utility shall
 102950 output a time exactly if it can be represented exactly as a decimal number, and otherwise shall
 102951 generate only enough digits so that the same time shall be recovered if the file is extracted on a
 102952 system whose underlying implementation supports the same time granularity.

102953 **ustar Interchange Format**

102954 A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a
 102955 fixed-size logical record of 512 octets (see below). Although this format may be thought of as
 102956 being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of
 102957 transportable media are not excluded. Each file archived shall be represented by a header logical
 102958 record that describes the file, followed by zero or more logical records that give the contents of
 102959 the file. At the end of the archive file there shall be two 512-octet logical records filled with
 102960 binary zeros, interpreted as an end-of-archive indicator.

102961 The logical records may be grouped for physical I/O operations, as described under the
 102962 **-bblocksize** and **-x ustar** options. Each group of logical records may be written with a single
 102963 operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a
 102964 single tape physical block. The last physical block shall always be the full size, so logical records
 102965 after the two zero logical records may contain undefined data.

102966 The header logical record shall be structured as shown in the following table. All lengths and
 102967 offsets are in decimal.

102968 **Table 4-14** ustar Header Block

Field Name	Octet Offset	Length (in Octets)
<i>name</i>	0	100
<i>mode</i>	100	8
<i>uid</i>	108	8
<i>gid</i>	116	8
<i>size</i>	124	12
<i>mtime</i>	136	12
<i>chksum</i>	148	8
<i>typeflag</i>	156	1
<i>linkname</i>	157	100
<i>magic</i>	257	6
<i>version</i>	263	2
<i>uname</i>	265	32
<i>gname</i>	297	32
<i>devmajor</i>	329	8
<i>devminor</i>	337	8
<i>prefix</i>	345	155

102986 All characters in the header logical record shall be represented in the coded character set of the
 102987 ISO/IEC 646: 1991 standard. For maximum portability between implementations, names should
 102988 be selected from characters represented by the portable filename character set as octets with the
 102989 most significant bit zero. If an implementation supports the use of characters outside of <slash>
 102990 and the portable filename character set in names for files, users, and groups, one or more
 102991 implementation-defined encodings of these characters shall be provided for interchange
 102992 purposes.

102993 However, the *pax* utility shall never create filenames on the local system that cannot be accessed

102994 via the procedures described in POSIX.1-2017. If a filename is found on the medium that would
 102995 create an invalid filename, it is implementation-defined whether the data from the file is stored
 102996 on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these
 102997 files as long as it produces an error indicating that the file is being ignored.

102998 Each field within the header logical record is contiguous; that is, there is no padding used. Each
 102999 character on the archive medium shall be stored contiguously.

103000 The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character.
 103001 The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all
 103002 characters in the array contain non-NUL characters including the last character. The *version* field
 103003 is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character.
 103004 All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991
 103005 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

103006 The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be
 103007 formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up
 103008 to the first NUL character), a <slash> character, and *name*; otherwise, *name* is used alone. In
 103009 either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it
 103010 shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a
 103011 pathname does not fit in the space provided, *pax* shall notify the user of the error, and shall not
 103012 store any part of the file †header or data †on the medium.

103013 The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a
 103014 *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall
 103015 notify the user of the error, and shall not attempt to store the link on the medium.

103016 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit
 103017 representation. The encoded bits shall represent the following values:

103018 **Table 4-15** *ustar mode* Field

Bit Value	POSIX.1-2017 Bit	Description
04 000	S_ISUID	Set UID on execution.
02 000	S_ISGID	Set GID on execution.
01 000	<reserved>	Reserved for future standardization.
00 400	S_IRUSR	Read permission for file owner class.
00 200	S_IWUSR	Write permission for file owner class.
00 100	S_IXUSR	Execute/search permission for file owner class.
00 040	S_IRGRP	Read permission for file group class.
00 020	S_IWGRP	Write permission for file group class.
00 010	S_IXGRP	Execute/search permission for file group class.
00 004	S_IROTH	Read permission for file other class.
00 002	S_IWOTH	Write permission for file other class.
00 001	S_IXOTH	Execute/search permission for file other class.

103032 When appropriate privileges are required to set one of these mode bits, and the user restoring
 103033 the files from the archive does not have appropriate privileges, the mode bits for which the user
 103034 does not have appropriate privileges shall be ignored. Some of the mode bits in the archive
 103035 format are not mentioned elsewhere in this volume of POSIX.1-2017. If the implementation does
 103036 not support those bits, they may be ignored.

103037 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

103038 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type
 103039 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to

- 103040 specify a file of type 5 (directory), the *size* field shall be interpreted as described under the
 103041 definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag*
 103042 field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size*
 103043 field is unspecified by this volume of POSIX.1-2017, and no data logical records shall be stored
 103044 on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the
 103045 *typeflag* field is set to any other value, the number of logical records written following the header
 103046 shall be $(size+511)/512$, ignoring any fraction in the result of the division.
- 103047 The *mtime* field shall be the modification time of the file at the time it was archived. It is the
 103048 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained
 103049 from the *stat()* function.
- 103050 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of
 103051 the simple sum of all octets in the header logical record. Each octet in the header shall be treated
 103052 as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the
 103053 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is
 103054 treated as if it were all <space> characters.
- 103055 The *typeflag* field specifies the type of file archived. If a particular implementation does not
 103056 recognize the type, or the user does not have appropriate privileges to create that type, the file
 103057 shall be extracted as if it were a regular file if the file type is defined to have a meaning for the
 103058 *size* field that could cause data logical records to be written on the medium (see the previous
 103059 description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error
 103060 indicating that the conversion took place. All of the *typeflag* fields shall be coded in the
 103061 ISO/IEC 646:1991 standard IRV:
- | | | |
|--------|------|--|
| 103062 | 0 | Represents a regular file. For backwards-compatibility, a <i>typeflag</i> value of binary zero (' <code>\0</code> ') should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a <i>typeflag</i> value of the ISO/IEC 646:1991 standard IRV ' <code>0</code> '. |
| 103063 | | |
| 103064 | | |
| 103065 | | |
| 103066 | 1 | Represents a file linked to another file, of any type, previously archived. Such files are identified by having the same device and file serial numbers, and pathnames that refer to different directory entries. All such files shall be archived as linked files. The linked-to name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than 100 octets in length. |
| 103067 | | |
| 103068 | | |
| 103069 | | |
| 103070 | | |
| 103071 | 2 | Represents a symbolic link. The contents of the symbolic link shall be stored in the <i>linkname</i> field. |
| 103072 | | |
| 103073 | 3, 4 | Represent character special files and block special files respectively. In this case the <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format of which is unspecified by this volume of POSIX.1-2017. Implementations may map the device specifications to their own local specification or may ignore the entry. |
| 103074 | | |
| 103075 | | |
| 103076 | | |
| 103077 | 5 | Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i> field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the <i>size</i> field. |
| 103078 | | |
| 103079 | | |
| 103080 | | |
| 103081 | | |
| 103082 | 6 | Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents. |
| 103083 | | |
| 103084 | 7 | Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0). |
| 103085 | | |
| 103086 | | |

103087 A-Z The letters 'A' to 'Z', inclusive, are reserved for custom implementations. All other
103088 values are reserved for future versions of this standard.

103089 It is unspecified whether files with pathnames that refer to the same directory entry are archived
103090 as linked files or as separate files. If they are archived as linked files, this means that attempting
103091 to extract both pathnames from the resulting archive will always cause an error (unless the `-u`
103092 option is used) because the link cannot be created.

103093 It is unspecified whether files with the same device and file serial numbers being appended to
103094 an archive are treated as linked files to members that were in the archive before the append.

103095 Attempts to archive a socket shall produce a diagnostic message when **ustar** interchange format
103096 is used, but may be allowed when **pax** interchange format is used. Handling of other file types is
103097 implementation-defined.

103098 The *magic* field is the specification that this archive was output in this archive format. If this field
103099 contains **ustar** (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by
103100 NUL), the *uname* and *gname* fields shall contain the ISO/IEC 646:1991 standard IRV
103101 representation of the owner and group of the file, respectively (truncated to fit, if necessary).
103102 When the file is restored by a privileged, protection-preserving version of the utility, the user
103103 and group databases shall be scanned for these names. If found, the user and group IDs
103104 contained within these files shall be used rather than the values contained within the *uid* and *gid*
103105 fields.

103106 **cpio Interchange Format**

103107 The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that
103108 describes the file, the name of the file, and then the contents of the file.

103109 An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used
103110 only to make physical I/O more efficient. The last group of blocks shall always be at the full
103111 size.

103112 For the octet-oriented **cpio** archive format, the individual entry information shall be in the order
103113 indicated and described by the following table; see also the **<cpio.h>** header.

103114

Table 4-16 Octet-Oriented cpio Archive Entry

103115

Header Field Name	Length (in Octets)	Interpreted as
<i>c_magic</i>	6	Octal number
<i>c_dev</i>	6	Octal number
<i>c_ino</i>	6	Octal number
<i>c_mode</i>	6	Octal number
<i>c_uid</i>	6	Octal number
<i>c_gid</i>	6	Octal number
<i>c_nlink</i>	6	Octal number
<i>c_rdev</i>	6	Octal number
<i>c_mtime</i>	11	Octal number
<i>c_namesize</i>	6	Octal number
<i>c_filesize</i>	11	Octal number
Filename Field Name	Length	Interpreted as
<i>c_name</i>	<i>c_namesize</i>	Pathname string
File Data Field Name	Length	Interpreted as
<i>c_filedata</i>	<i>c_filesize</i>	Data

103116

103117

103118

103119

103120

103121

103122

103123

103124

103125

103126

103127

103128

103129

103130

103131

cpio Header

103132

103133

103134

103135

103136

103137

For each file in the archive, a header as defined previously shall be written. The information in the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending the ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result is written to the most-significant digit of the stream of octets first. The fields shall be interpreted as follows:

103138

103139

c_magic Identify the archive as being a transportable archive by containing the identifying value "070707".

103140

103141

103142

c_dev, c_ino Contains values that uniquely identify the file within the archive (that is, no files contain the same pair of *c_dev* and *c_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.

103143

c_mode Contains the file type and access permissions as defined in the following table.

103144

Table 4-17 Values for `cpio c_mode` Field

103145

103146

103147

103148

103149

103150

103151

103152

103153

103154

103155

103156

103157

103158

103159

103160

103161

103162

103163

103164

103165

103166

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set <i>uid</i>
C_ISGID	002 000	Set <i>gid</i>
C_ISVTX	001 000	Reserved
File Type Name	Value	Indicates
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISLNK	0120 000	Symbolic link
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISSOCK	0140 000	Socket
C_ISCTG	0110 000	Reserved

103167

103168

103169

103170

103171

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of POSIX.1-2017; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

103172

c_uid

Contains the user ID of the owner.

103173

c_gid

Contains the group ID of the group.

103174

c_nlink

103175

103176

103177

103178

103179

Contains a number greater than or equal to the number of links in the archive referencing the file. If the `-a` option is used to append to a *cpio* archive, then the *pax* utility need not account for the files in the existing part of the archive when calculating the *c_nlink* values for the appended part of the archive, and need not alter the *c_nlink* values in the existing part of the archive if additional files with the same *c_dev* and *c_ino* values are appended to the archive.

103180

c_rdev

Contains implementation-defined information for character or block special files.

103181

c_mtime

103182

Contains the latest time of modification of the file at the time the archive was created.

103183

c_namesize

Contains the length of the pathname, including the terminating NUL character.

103184

c_filesize

Contains the length in octets of the data section following the header structure.

103185 **cpio Filename**

103186 The *c_name* field shall contain the pathname of the file. The length of this field in octets is the
103187 value of *c_namesize*.

103188 If a filename is found on the medium that would create an invalid pathname, it is
103189 implementation-defined whether the data from the file is stored on the file hierarchy and under
103190 what name it is stored.

103191 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum
103192 portability between implementations, names should be selected from characters represented by
103193 the portable filename character set as octets with the most significant bit zero. If an
103194 implementation supports the use of characters outside the portable filename character set in
103195 names for files, users, and groups, one or more implementation-defined encodings of these
103196 characters shall be provided for interchange purposes. However, the *pax* utility shall never create
103197 filenames on the local system that cannot be accessed via the procedures described previously in
103198 this volume of POSIX.1-2017. If a filename is found on the medium that would create an invalid
103199 filename, it is implementation-defined whether the data from the file is stored on the local file
103200 system and under what name it is stored. The *pax* utility may choose to ignore these files as long
103201 as it produces an error indicating that the file is being ignored.

103202 **cpio File Data**

103203 Following *c_name*, there shall be *c_filesiz*e octets of data. Interpretation of such data occurs in a
103204 manner dependent on the file. For regular files, the data shall consist of the contents of the file.
103205 For symbolic links, the data shall consist of the contents of the symbolic link. If *c_filesiz*e is zero,
103206 no data shall be contained in *c_filedata*.

103207 When restoring from an archive:

103208 If the user does not have appropriate privileges to create a file of the specified type, *pax*
103209 shall ignore the entry and write an error message to standard error.

103210 Only regular files and symbolic links have data to be restored. Presuming a regular file
103211 meets any selection criteria that might be imposed on the format-reading utility by the
103212 user, such data shall be restored.

103213 If a user does not have appropriate privileges to set a particular mode flag, the flag shall be
103214 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this
103215 volume of POSIX.1-2017. If the implementation does not support those flags, they may be
103216 ignored.

103217 **cpio Special Entries**

103218 FIFO special files, directories, and the trailer shall be recorded with *c_filesiz*e equal to zero.
103219 Symbolic links shall be recorded with *c_filesiz*e equal to the length of the contents of the symbolic
103220 link. For other special files, *c_filesiz*e is unspecified by this volume of POSIX.1-2017. The header
103221 for the next file entry in the archive shall be written directly after the last octet of the file entry
103222 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive;
103223 the contents of octets in the last block of the archive following such a header are undefined.

103224 **EXIT STATUS**

103225 The following exit values shall be returned:

103226 0 All files were processed successfully.

103227 >0 An error occurred.

103228 CONSEQUENCES OF ERRORS

103229 If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an
 103230 archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a
 103231 diagnostic message shall be written to standard error and a non-zero exit status shall be
 103232 returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax*
 103233 shall not, by default, create a second copy of the file.

103234 If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may
 103235 have only partially extracted the file or (if the **-n** option was not specified) may have extracted a
 103236 file of the same name as that specified by the user, but which is not the file the user wanted.
 103237 Additionally, the file modes of extracted directories may have additional bits from the S_IRWXU
 103238 mask set as well as incorrect modification and access times.

103239 APPLICATION USAGE

103240 Caution is advised when using the **-a** option to append to a *cpio* format archive. If any of the
 103241 files being appended happen to be given the same *c_dev* and *c_ino* values as a file in the existing
 103242 part of the archive, then they may be treated as links to that file on extraction. Thus, it is risky to
 103243 use **-a** with *cpio* format except when it is done on the same system that the original archive was
 103244 created on, and with the same *pax* utility, and in the knowledge that there has been little or no
 103245 file system activity since the original archive was created that could lead to any of the files
 103246 appended being given the same *c_dev* and *c_ino* values as an unrelated file in the existing part of
 103247 the archive. Also, when (intentionally) appending additional links to a file in the existing part of
 103248 the archive, the *c_nlink* values in the modified archive can be smaller than the number of links to
 103249 the file in the archive, which may mean that the links are not preserved on extraction.

103250 The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio*
 103251 implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p**
 103252 option also provides a consistent means of extending the ways in which future file attributes can
 103253 be addressed, such as for enhanced security systems or high-performance files. Although it may
 103254 seem complex, there are really two modes that are most commonly used:

103255 **-p e** “Preserve everything”. This would be used by the historical superuser, someone with
 103256 all appropriate privileges, to preserve all aspects of the files as they are recorded in the
 103257 archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined attributes.

103258 **-p p** “Preserve” the file mode bits. This would be used by the user with regular privileges
 103259 who wished to preserve aspects of the file other than the ownership. The file times are
 103260 preserved by default, but two other flags are offered to disable these and use the time
 103261 of extraction.

103262 The one pathname per line format of standard input precludes pathnames containing <newline>
 103263 characters. Although such pathnames violate the portable filename guidelines, they may exist
 103264 and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from
 103265 historical archive programs. The problem can be avoided by listing filename arguments on the
 103266 command line instead of on standard input.

103267 It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this
 103268 volume of POSIX.1-2017. Specifically, creating files of type block special or character special,
 103269 restoring file access times unless the files are owned by the user (the **-t** option), or preserving file
 103270 owner, group, and mode (the **-p** option) all probably require appropriate privileges.

103271 In **read** mode, implementations are permitted to overwrite files when the archive has multiple
 103272 members with the same name. This may fail if permissions on the first version of the file do not
 103273 permit it to be overwritten.

103274 The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes ($8 * 2^{30}$) in size.

103275 When archives containing binary header information are listed , the filenames printed may
103276 cause strange behavior on some terminals.

103277 When all of the following are true:

- 103278 1. A file of type directory is being placed into an archive.
- 103279 2. The **ustar** archive format is being used.
- 103280 3. The pathname of the directory is less than or equal to 155 bytes long (it will fit in the *prefix*
103281 field in the **ustar** header block).
- 103282 4. The last component of the pathname of the directory is longer than 100 bytes long (it will
103283 not fit in the *name* field in the **ustar** header block).

103284 some implementations of the *pax* utility will place the entire directory pathname in the *prefix*
103285 field, set the *name* field to an empty string, and place the directory in the archive. Other
103286 implementations of the *pax* utility will give an error under these conditions because the *name*
103287 field is not large enough to hold the last component of the directory name. This standard allows
103288 either behavior. However, when extracting a directory from a **ustar** format archive, this standard
103289 requires that all implementations be able to extract a directory even if the *name* field contains an
103290 empty string as long as the *prefix* field does not also contain an empty string.

103291 **EXAMPLES**

103292 The following command:

```
103293 pax -w -f /dev/rmt/1m .
```

103294 copies the contents of the current directory to tape drive 1, medium density (assuming historical
103295 System V device naming procedures—the historical BSD device name would be **/dev/rmt9**).

103296 The following commands:

```
103297 mkdir newdir
103298 pax -rw olddir newdir
```

103299 copy the *olddir* directory hierarchy to *newdir*.

```
103300 pax -r -s ',^//*usr//*,,' -f a.pax
```

103301 reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current
103302 directory.

103303 Using the option:

```
103304 -o listopt="%M %(atime)T %(size)D %(name)s"
```

103305 overrides the default output description in Standard Output and instead writes:

```
103306 -rw-rw--- Jan 12 15:53 2003 1492 /usr/foo/bar
```

103307 Using the options:

```
103308 -o listopt='%L\t%(size)D\n%.7' \
103309 -o listopt='(name)s\n%(atime)T\n%T'
```

103310 overrides the default output description in Standard Output and instead writes:

```
103311 /usr/foo/bar -> /tmp 1492
103312 /usr/fo
103313 Jan 12 15:53 1991
```

103314 Jan 31 15:53 2003

103315 **RATIONALE**

103316 The *pax* utility was new for the ISO POSIX-2:1993 standard. It represents a peaceful compromise
103317 between advocates of the historical *tar* and *cpio* utilities.

103318 A fundamental difference between *cpio* and *tar* was in the way directories were treated. The *cpio*
103319 utility did not treat directories differently from other files, and to select a directory and its
103320 contents required that each file in the hierarchy be explicitly specified. For *tar*, a directory
103321 matched every file in the file hierarchy it rooted.

103322 The *pax* utility offers both interfaces; by default, directories map into the file hierarchy they root.
103323 The `-d` option causes *pax* to skip any file not explicitly referenced, as *cpio* historically did. The *tar*
103324 `-style` behavior was chosen as the default because it was believed that this was the more
103325 common usage and because *tar* is the more commonly available interface, as it was historically
103326 provided on both System V and BSD implementations.

103327 The data interchange format specification in this volume of POSIX.1-2017 requires that processes
103328 with “appropriate privileges” shall always restore the ownership and permissions of extracted
103329 files exactly as archived. If viewed from the historic equivalence between superuser and
103330 “appropriate privileges”, there are two problems with this requirement. First, users running as
103331 superusers may unknowingly set dangerous permissions on extracted files. Second, it is
103332 needlessly limiting, in that superusers cannot extract files and own them as superuser unless the
103333 archive was created by the superuser. (It should be noted that restoration of ownerships and
103334 permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to
103335 avoid these two problems, the *pax* specification has an additional “privilege” mechanism, the `-p`
103336 option. Only a *pax* invocation with the privileges needed, and which has the `-p` option set using
103337 the `e` specification character, has appropriate privileges to restore full ownership and permission
103338 information.

103339 Note also that this volume of POSIX.1-2017 requires that the file ownership and access
103340 permissions shall be set, on extraction, in the same fashion as the *creat()* function when provided
103341 with the mode stored in the archive. This means that the file creation mask of the user is applied
103342 to the file permissions.

103343 Users should note that directories may be created by *pax* while extracting files with permissions
103344 that are different from those that existed at the time the archive was created. When extracting
103345 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set
103346 their file creation mask appropriately to protect these files during extraction.

103347 The table of contents output is written to standard output to facilitate pipeline processing.

103348 An early proposal had hard links displaying for all pathnames. This was removed because it
103349 complicates the output of the case where `-v` is not specified and does not match historical *cpio*
103350 usage. The hard-link information is available in the `-v` display.

103351 The description of the `-l` option allows implementations to make hard links to symbolic links.
103352 Earlier versions of this standard did not specify any way to create a hard link to a symbolic link,
103353 but many implementations provided this capability as an extension. If there are hard links to
103354 symbolic links when an archive is created, the implementation is required to archive the hard
103355 link in the archive (unless `-H` or `-L` is specified). When in **read** mode and in **copy** mode,
103356 implementations supporting hard links to symbolic links should use them when appropriate.

103357 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have
103358 been brought along from historical usage. For example, there are restrictions on the length of
103359 pathnames stored in the archive. When *pax* is used in **copy(-rw)** mode (copying directory
103360 hierarchies), the ability to use extensions from the `-xpax` format overcomes these restrictions.

103361 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard
103362 block-size values for *cpio*, set when the **-B** option is specified. (The other default block-size value
103363 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240
103364 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The
103365 maximum block size of 32 256 bytes (2^{15} -512 bytes) is the largest multiple of 512 bytes that fits
103366 into a signed 16-bit tape controller transfer register. There are known limitations in some
103367 historical systems that would prevent larger blocks from being accepted. Historical values were
103368 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate
103369 archives. Also, default block sizes for any file type other than character special file has been
103370 deleted from this volume of POSIX.1-2017 as unimportant and not likely to affect the structure of
103371 the resulting archive.

103372 Implementations are permitted to modify the block-size value based on the archive format or the
103373 device to which the archive is being written. This is to provide implementations with the
103374 opportunity to take advantage of special types of devices, and it should not be used without a
103375 great deal of consideration as it almost certainly decreases archive portability.

103376 The intended use of the **-n** option was to permit extraction of one or more files from the archive
103377 without processing the entire archive. This was viewed by the standard developers as offering
103378 significant performance advantages over historical implementations. The **-n** option in early
103379 proposals had three effects; the first was to cause special characters in patterns to not be treated
103380 specially. The second was to cause only the first file that matched a pattern to be extracted. The
103381 third was to cause *pax* to write a diagnostic message to standard error when no file was found
103382 matching a specified pattern. Only the second behavior is retained by this volume of
103383 POSIX.1-2017, for many reasons. First, it is in general not acceptable for a single option to have
103384 multiple effects. Second, the ability to make pattern matching characters act as normal characters
103385 is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the
103386 special characters is useful because users may wish to normalize only a single special character
103387 in a single filename. Fourth, given a more general escape mechanism, the previous behavior of
103388 the **-n** option can be easily obtained using the **-s** option or a *sed* script. Finally, writing a
103389 diagnostic message when a pattern specified by the user is unmatched by any file is useful
103390 behavior in all cases.

103391 In this version, the **-n** was removed from the **copy** mode synopsis of *pax*; it is inapplicable
103392 because there are no pattern operands specified in this mode.

103393 There is another method than *pax* for copying subtrees in POSIX.1-2017 described as part of the
103394 *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface,
103395 while *pax* offers a finer granularity of control. Each provides additional functionality to the
103396 other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is
103397 the intention of the standard developers that the results be similar (using appropriate option
103398 combinations in both utilities). The results are not required to be identical; there seemed
103399 insufficient gain to applications to balance the difficulty of implementations having to guarantee
103400 that the results would be exactly identical.

103401 A single archive may span more than one file. It is suggested that implementations provide
103402 informative messages to the user on standard error whenever the archive file is changed.

103403 The **-d** option (do not create intermediate directories not listed in the archive) found in early
103404 proposals was originally provided as a complement to the historic **-d** option of *cpio*. It has been
103405 deleted.

103406 The **-s** option in early proposals specified a subset of the substitution command from the *ed*
103407 utility. As there was no reason for only a subset to be supported, the **-s** option is now compatible
103408 with the current *ed* specification. Since the delimiter can be any non-null character, the following

103409 usage with single <space> characters is valid:

```
103410 pax -s " foo bar " ...
```

103411 The **-t** description is worded so as to note that this may cause the access time update caused by
103412 some other activity (which occurs while the file is being read) to be overwritten.

103413 The default behavior of *pax* with regard to file modification times is the same as historical
103414 implementations of *tar*. It is not the historical behavior of *cpio*.

103415 Because the **-i** option uses **/dev/tty**, utilities without a controlling terminal are not able to use
103416 this option.

103417 The **-y** option, found in early proposals, has been deleted because a line containing a single
103418 <period> for the **-i** option has equivalent functionality. The special lines for the **-i** option (a
103419 single <period> and the empty line) are historical practice in *cpio*.

103420 In early drafts, a **-echarmap** option was included to increase portability of files between systems
103421 using different coded character sets. This option was omitted because it was apparent that
103422 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate
103423 substitute.

103424 The ISO POSIX-2:1993 standard and ISO POSIX-1 standard requirements for *pax*, however,
103425 made it very difficult to create a single archive containing files created using extended characters
103426 provided by different locales. This version adds the **hdrcharset** keyword to make it possible to
103427 archive files in these cases without dropping files due to translation errors.

103428 Translating filenames and other attributes from a locale's encoding to UTF-8 and then back again
103429 can lose information, as the resulting filename might not be byte-for-byte equivalent to the
103430 original. To avoid this problem, users can specify the **-o hdrcharset=binary** option, which will
103431 cause the resulting archive to use binary format for all names and attributes. Such archives are
103432 not portable among hosts that use different native encodings (e.g., EBCDIC *versus* ASCII-based
103433 encodings), but they will allow interchange among the vast majority of POSIX file systems in
103434 practical use. Also, the **-o hdrcharset=binary** option will cause *pax* in **copy** mode to behave
103435 more like other standard utilities such as *cp*.

103436 If the values specified by the **-o exthdr.name=value**, **-o globexthdr.name=value**, or by
103437 **\$TMPDIR** (if **-o globexthdr.name** is not specified) require a character encoding other than that
103438 described in the ISO/IEC 646:1991 standard, a **path** extended header record will have to be
103439 created for the file. If a **hdrcharset** extended header record is active for such headers, it will
103440 determine the codeset used for the value field in these extended **path** header records. These **path**
103441 extended header records always need to be created when writing an archive even if
103442 **hdrcharset=binary** has been specified and would contain the same (binary) data that appears in
103443 the **ustar** header record prefix and *name* fields. (In other words, an extended header **path** record
103444 is always required to be generated if the *prefix* or *name* fields contain non-ASCII characters even
103445 when **hdrcharset=binary** is also in effect for that file.)

103446 The **-k** option was added to address international concerns about the dangers involved in the
103447 character set transformations of **-e** (if the target character set were different from the source, the
103448 filenames might be transformed into names matching existing files) and also was made more
103449 general to protect files transferred between file systems with different {NAME_MAX} values
103450 (truncating a filename on a smaller system might also inadvertently overwrite existing files). As
103451 stated, it prevents any overwriting, even if the target file is older than the source. This version
103452 adds more granularity of options to solve this problem by introducing the **-o invalid=option** ‡
103453 specifically the **UTF-8** and **binary** actions. (Note that an existing file is still subject to overwriting
103454 in this case. The **-k** option closes that loophole.)

103455 Some of the file characteristics referenced in this volume of POSIX.1-2017 might not be
 103456 supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the
 103457 file access time. For this reason, the **e** specification character has been provided, intended to
 103458 cause all file characteristics specified in the archive to be retained.

103459 It is required that extracted directories, by default, have their access and modification times and
 103460 permissions set to the values specified in the archive. This has obvious problems in that the
 103461 directories are almost certainly modified after being extracted and that directory permissions
 103462 may not permit file creation. One possible solution is to create directories with the mode
 103463 specified in the archive, as modified by the *umask* of the user, with sufficient permissions to
 103464 allow file creation. After all files have been extracted, *pax* would then reset the access and
 103465 modification times and permissions as necessary.

103466 The list-mode formatting description borrows heavily from the one defined by the *printf* utility.
 103467 However, since there is no separate operand list to get conversion arguments, the format was
 103468 extended to allow specifying the name of the conversion argument as part of the conversion
 103469 specification.

103470 The **T** conversion specifier allows time fields to be displayed in any of the date formats. Unlike
 103471 the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past.
 103472 This makes parsing the output more predictable.

103473 The **D** conversion specifier handles the ability to display the major/minor or file size, as with *ls*,
 103474 by using `%-8(size)D`.

103475 The **L** conversion specifier handles the *ls* display for symbolic links.

103476 Conversion specifiers were added to generate existing known types used for *ls*.

103477 **pax Interchange Format**

103478 The new POSIX data interchange format was developed primarily to satisfy international
 103479 concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded
 103480 in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers
 103481 realized that this new POSIX data interchange format should be very extensible because there
 103482 were other requirements they foresaw in the near future:

103483 Support international character encodings and locale information

103484 Support security information (ACLs, and so on)

103485 Support future file types, such as realtime or contiguous files

103486 Include data areas for implementation use

103487 Support systems with words larger than 32 bits and timers with subsecond granularity

103488 The following were not goals for this format because these are better handled by separate
 103489 utilities or are inappropriate for a portable format:

103490 Encryption

103491 Compression

103492 Data translation between locales and codesets

103493 *inode* storage

103494 The format chosen to support the goals is an extension of the **ustar** format. Of the two formats
 103495 previously available, only the **ustar** format was selected for extensions because:

103496 It was easier to extend in an upwards-compatible way. It offered version flags and header
103497 block type fields with room for future standardization. The **cpio** format, while possessing a
103498 more flexible file naming methodology, could not be extended without breaking some
103499 theoretical implementation or using a dummy filename that could be a legitimate filename.

103500 Industry experience since the original “tar wars” fought in developing the ISO POSIX-1
103501 standard has clearly been in favor of the **ustar** format, which is generally the default
103502 output format selected for *pax* implementations on new systems.

103503 The new format was designed with one additional goal in mind: reasonable behavior when an
103504 older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated
103505 that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this
103506 allowed the format to include all the extended information in a pseudo-regular file that
103507 preceded each real file. An option is given that allows the archive creator to set up reasonable
103508 names for these files on the older systems. Also, the normative text suggests that reasonable file
103509 access values be used for this **ustar** header block. Making these header files inaccessible for
103510 convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are
103511 suggested.

103512 The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format
103513 rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous
103514 version of *pax*), mandated the behavior of the format-reading utility when it encountered an
103515 unknown *typeflag*, but was silent about the other two fields.

103516 Early proposals for the first version of this standard contained a proposed archive format that
103517 was based on compatibility with the standard for tape files (ISO 1001, similar to the format used
103518 historically on many mainframes and minicomputers). This format was overly complex and
103519 required considerable overhead in volume and header records. Furthermore, the standard
103520 developers felt that it would not be acceptable to the community of POSIX developers, so it was
103521 later changed to be a format more closely related to historical practice on POSIX systems.

103522 The prefix and name split of pathnames in **ustar** was replaced by the single path extended
103523 header record for simplicity.

103524 The concept of a global extended header (*typeflag g*) was controversial. If this were applied to an
103525 archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape
103526 could be a serious problem; a utility attempting to extract as many files as possible from a
103527 damaged archive could lose a large percentage of file header information in this case. However,
103528 if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers
103529 considerable potential size reductions by eliminating redundant information. Thus, the text
103530 warns against using the global method for unreliable media and provides a method for
103531 implanting global information in the extended header for each file, rather than in the *typeflag g*
103532 records.

103533 No facility for data translation or filtering on a per-file basis is included because the standard
103534 developers could not invent an interface that would allow this in an efficient manner. If a filter,
103535 such as encryption or compression, is to be applied to all the files, it is more efficient to apply the
103536 filter to the entire archive as a single file. The standard developers considered interfaces that
103537 would invoke a shell script for each file going into or out of the archive, but the system overhead
103538 in this approach was considered to be too high.

103539 One such approach would be to have **filter=** records that give a pathname for an executable.
103540 When the program is invoked, the file and archive would be open for standard input/output
103541 and all the header fields would be available as environment variables or command-line
103542 arguments. The standard developers did discuss such schemes, but they were omitted from
103543 POSIX.1-2017 due to concerns about excessive overhead. Also, the program itself would need to

103544 be in the archive if it were to be used portably.

103545 There is currently no portable means of identifying the character set(s) used for a file in the file
103546 system. Therefore, *pax* has not been given a mechanism to generate charset records
103547 automatically. The only portable means of doing this is for the user to write the archive using the
103548 **-ocharset=string** command line option. This assumes that all of the files in the archive use the
103549 same encoding. The "implementation-defined" text is included to allow for a system that can
103550 identify the encodings used for each of its files.

103551 The table of standards that accompanies the charset record description is acknowledged to be
103552 very limited. Only a limited number of character set standards is reasonable for maximal
103553 interchange. Any character set is, of course, possible by prior agreement. It was suggested that
103554 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal
103555 standards, and then only those with reasonably large followings, can be included here, simply as
103556 a matter of practicality. The *<value>*s represent names of officially registered character sets in the
103557 format required by the ISO 2375:1985 standard.

103558 The normal *<comma>* or *<blank>*-separated list rules are not followed in the case of keyword
103559 options to allow ease of argument parsing for *getopts*.

103560 Further information on character encodings is in [pax Archive Character Set Encoding/Decoding](#)
103561 (on page 3102).

103562 The standard developers have reserved keyword name space for vendor extensions. It is
103563 suggested that the format to be used is:

103564 *VENDOR.keyword*

103565 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further
103566 suggested that the keyword following the *<period>* be named differently than any of the
103567 standard keywords so that it could be used for future standardization, if appropriate, by
103568 omitting the *VENDOR* prefix.

103569 The *<length>* field in the extended header record was included to make it simpler to step
103570 through the records, even if a record contains an unknown format (to a particular *pax*) with
103571 complex interactions of special characters. It also provides a minor integrity checkpoint within
103572 the records to aid a program attempting to recover files from a damaged archive.

103573 There are no extended header versions of the *devmajor* and *devminor* fields because the
103574 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific
103575 extended keywords (such as *VENDOR.devmajor*) should be used.

103576 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly
103577 on a symbolic name basis, as in **ustar**.

103578 Just as with the **ustar** format descriptions, the new format makes no special arrangements for
103579 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file
103580 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing
103581 their labels, and mounting each in the proper sequence are considered to be implementation
103582 details that cannot be described portably.

103583 The **pax** format is intended for interchange, not only for backup on a single (family of) systems.
103584 It is not as densely packed as might be possible for backup:

103585 It contains information as coded characters that could be coded in binary.

103586 It identifies extended records with name fields that could be omitted in favor of a fixed-
103587 field layout.

103588 It translates names into a portable character set and identifies locale-related information,
103589 both of which are probably unnecessary for backup.

103590 The requirements on restoring from an archive are slightly different from the historical wording,
103591 allowing for non-monolithic privilege to bring forward as much as possible. In particular,
103592 attributes such as “high performance file” might be broadly but not universally granted while
103593 set-user-ID or *chown()* might be much more restricted. There is no implication in POSIX.1-2017
103594 that the security information be honored after it is restored to the file hierarchy, in spite of what
103595 might be improperly inferred by the silence on that topic. That is a topic for another standard.

103596 Links are recorded in the fashion described here because a link can be to any file type. It is
103597 desirable in general to be able to restore part of an archive selectively and restore all of those files
103598 completely. If the data is not associated with each link, it is not possible to do this. However, the
103599 data associated with a file can be large, and when selective restoration is not needed, this can be
103600 a significant burden. The archive is structured so that files that have no associated data can
103601 always be restored by the name of any link name of any link, and the user may choose whether
103602 data is recorded with each instance of a file that contains data. The format permits mixing of
103603 both types of links in a single archive; this can be done for special needs, and *pax* is expected to
103604 interpret such archives on input properly, despite the fact that there is no *pax* option that would
103605 force this mixed case on output. (When **-o linkdata** is used, the output must contain the
103606 duplicate data, but the implementation is free to include it or omit it when **-o linkdata** is not
103607 used.)

103608 The time values are included as extended header records for those implementations needing
103609 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be
103610 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject
103611 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a
103612 leading '-'. Even though some implementations can support finer file-time granularities than
103613 seconds, the normative text requires support only for seconds since the Epoch because the
103614 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new
103615 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will
103616 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification
103617 time) is described with appropriate privileges so that it can be ignored when writing to the file
103618 system. POSIX does not provide a portable means to change file creation time. Nothing is
103619 intended to prevent a non-portable implementation of *pax* from restoring the value.

103620 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the
103621 sizes specified in the regular *tar* header. New file system architectures are emerging that will
103622 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits
103623 for user and group IDs, but the extended header values were included for completeness,
103624 allowing overrides for all of the decimal values in the *tar* header.

103625 The standard developers intended to describe the effective results of *pax* with regard to file
103626 ownerships and permissions; implementations are not restricted in timing or sequencing the
103627 restoration of such, provided the results are as specified.

103628 Much of the text describing the extended headers refers to use in “**write** or **copy** modes”. The
103629 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the
103630 copied files were written to an archive file and then subsequently extracted ...”. There is
103631 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,
103632 but the effects must be as if it had.

103633 **pax Archive Character Set Encoding/Decoding**

103634 There is a need to exchange archives of files between systems of different native codesets.
103635 Filenames, group names, and user names must be preserved to the fullest extent possible when
103636 an archive is read on the receiving platform. Translation of the contents of files is not within the
103637 scope of the *pax* utility.

103638 There will also be the need to represent characters that are not available on the receiving
103639 platform. These unsupported characters cannot be automatically folded to the local set of
103640 characters due to the chance of collisions. This could result in overwriting previous extracted
103641 files from the archive or pre-existing files on the system.

103642 For these reasons, the codeset used to represent characters within the extended header records of
103643 the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields
103644 requiring translation include, at a minimum, filenames, user names, group names, and link
103645 pathnames. Implementations may wish to have localized extended keywords that use non-
103646 portable characters.

103647 The standard developers considered the following options:

103648 The archive creator specifies the well-defined name of the source codeset. The receiver
103649 must then recognize the codeset name and perform the appropriate translations to the
103650 destination codeset.

103651 The archive creator includes within the archive the character mapping table for the source
103652 codeset used to encode extended header records. The receiver must then read the
103653 character mapping table and perform the appropriate translations to the destination
103654 codeset.

103655 The archive creator translates the extended header records in the source codeset into a
103656 canonical form. The receiver must then perform the appropriate translations to the
103657 destination codeset.

103658 The approach that incorporates the name of the source codeset poses the problem of codeset
103659 name registration, and makes the archive useless to *pax* archive decoders that do not recognize
103660 that codeset.

103661 Because parts of an archive may be corrupted, the standard developers felt that including the
103662 character map of the source codeset was too fragile. The loss of this one key component could
103663 result in making the entire archive useless. (The difference between this and the global extended
103664 header decision was that the latter has a workaround—duplicating extended header records on
103665 unreliable media—but this would be too burdensome for large character set maps.)

103666 Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the
103667 cross-product of all source and destination codesets.

103668 To simplify the translation from the source codeset to the canonical form and from the canonical
103669 form to the destination codeset, the standard developers decided that the internal representation
103670 should be a stateless encoding. A stateless encoding is one where each codepoint has the same
103671 meaning, without regard to the decoder being in a specific state. An example of a stateful
103672 encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the
103673 ISO/IEC 646: 1991 standard (equivalent to 7-bit ASCII).

103674 For these reasons, the standard developers decided to adopt a canonical format for the
103675 representation of file information strings. The obvious, well-endorsed candidate is the
103676 ISO/IEC 10646-1: 2000 standard (based in part on Unicode), which can be used to represent the
103677 characters of virtually all standardized character sets. The standard developers initially agreed
103678 upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters

103679 provides a sufficiently rich set to represent all commonly-used codesets.

103680 However, the standard developers found that the 16-bit Unicode representation had some
 103681 problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character
 103682 made the extended header records twice as long for the case of strings coded entirely from
 103683 historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the
 103684 ISO/IEC 10646-1:2000 standard. This multi-byte representation encodes UCS2 or UCS4
 103685 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In
 103686 addition, NUL octets and other characters possibly confusing to POSIX file systems do not
 103687 appear, except to represent themselves. It was realized that certain national codesets take up
 103688 more space after the encoding, due to their placement within the UCS range; it was felt that the
 103689 usefulness of the encoding of the names outweighs the disadvantage of size increase for file,
 103690 user, and group names.

103691 The encoding of UTF-8 is as follows:

103692	UCS4 Hex Encoding	UTF-8 Binary Encoding
103693	00000000–0000007F	0xxxxxxx
103694	00000080–000007FF	110xxxxx 10xxxxxx
103695	00000800–0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
103696	00010000–001FFFFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
103697	00200000–03FFFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
103698	04000000–7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

103699 where each 'x' represents a bit value from the character being translated.

103700 **ustar Interchange Format**

103701 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of
 103702 the historical *tar* utility. The goal of these changes was not only to provide the functional
 103703 enhancements desired, but also to retain compatibility between new and old versions. This
 103704 compatibility has been retained. Archives written using the old archive format are compatible
 103705 with the new format.

103706 Implementors should be aware that the previous file format did not include a mechanism to
 103707 archive directory type files. For this reason, the convention of using a filename ending with
 103708 <slash> was adopted to specify a directory on the archive.

103709 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for
 103710 {PATH_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname
 103711 be stored there without the use of the *prefix* field. Although the name field is known to be too
 103712 small to contain {PATH_MAX} characters, the value was not changed in this version of the
 103713 archive file format to retain backwards-compatibility, and instead the prefix was introduced.
 103714 Also, because of the earlier version of the format, there is no way to remove the restriction on the
 103715 *linkname* field being limited in size to just that of the *name* field.

103716 The *size* field is required to be meaningful in all implementation extensions, although it could be
 103717 zero. This is required so that the data blocks can always be properly counted.

103718 It is suggested that if device special files need to be represented that cannot be represented in the
 103719 standard format, that one of the extension types (A-Z) be used, and that the additional
 103720 information for the special file be represented as data and be reflected in the *size* field.

103721 Attempting to restore a special file type, where it is converted to ordinary data and conflicts with
 103722 an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax*
 103723 should not be able to overwrite the entries in, for example, */dev* in any case (whether the file is

103724 converted to another type or not). If run as a privileged user, it should be able to do so, and it
 103725 would be considered a bug if it did not. The same is true of ordinary data files and similarly
 103726 named special files; it is impossible to anticipate the needs of the user (who could really intend
 103727 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the
 103728 protection system as required.

103729 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a
 103730 **ustar** archive. POSIX.1-2017 does not require the contiguous file extension, but does define a
 103731 standard way of archiving such files so that all conforming systems can interpret these file types
 103732 in a meaningful and consistent manner. On a system that does not support extended file types,
 103733 the *pax* utility should do the best it can with the file and go on to the next.

103734 The file protection modes are those conventionally used by the *ls* utility. This is extended beyond
 103735 the usage in the ISO POSIX-2 standard to support the “shared text” or “sticky” bit. It is intended
 103736 that the conformance document should not document anything beyond the existence of and
 103737 support of such a mode. Further extensions are expected to these bits, particularly with
 103738 overloading the set-user-ID and set-group-ID flags.

103739 **cpio Interchange Format**

103740 The reference to appropriate privileges in the **cpio** format refers to an error on standard output;
 103741 the **ustar** format does not make comparable statements.

103742 The model for this format was the historical System V *cpio-c* data interchange format. This
 103743 model documents the portable version of the **cpio** format and not the binary version. It has the
 103744 flexibility to transfer data of any type described within POSIX.1-2017, yet is extensible to transfer
 103745 data types specific to extensions beyond POSIX.1-2017 (for example, contiguous files). Because it
 103746 describes existing practice, there is no question of maintaining upwards-compatibility.

103747 **cpio Header**

103748 There has been some concern that the size of the *c_ino* field of the header is too small to handle
 103749 those systems that have very large *inode* numbers. However, the *c_ino* field in the header is used
 103750 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as
 103751 the *inode* number of the file in the location from which that file is extracted.

103752 The name *c_magic* is based on historical usage.

103753 **cpio Filename**

103754 For most historical implementations of the *cpio* utility, {PATH_MAX} octets can be used to
 103755 describe the pathname without the addition of any other header fields (the NUL character
 103756 would be included in this count). {PATH_MAX} is the minimum value for pathname size,
 103757 documented as 256 bytes. However, an implementation may use *c_namesize* to determine the
 103758 exact length of the pathname. With the current description of the **<cpio.h>** header, this
 103759 pathname size can be as large as a number that is described in six octal digits.

103760 Two values are documented under the *c_mode* field values to provide for extensibility for known
 103761 file types:

103762 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the
 103763 information for this archive like a regular file. If this file type is undefined, the
 103764 implementation may create the file as a regular file.

103765 This provides for extensibility of the **cpio** format while allowing for the ability to read old
 103766 archives. Files of an unknown type may be read as “regular files” on some implementations. On
 103767 a system that does not support extended file types, the *pax* utility should do the best it can with

103768 the file and go on to the next.

103769 **FUTURE DIRECTIONS**

103770 None.

103771 **SEE ALSO**

103772 [Chapter 2](#) (on page 2345), *cp*, *ed*, *getopts*, *ls*, *printf*

103773 XBD [Section 3.169](#) (on page 60), [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#)
103774 (on page 216), [<cpio.h>](#), [<tar.h>](#)

103775 XSH *chown()*, *creat()*, *fstatat()*, *mkdir()*, *mkfifo()*, *utime()*, *write()*

103776 **CHANGE HISTORY**

103777 First released in Issue 4.

103778 **Issue 5**

103779 A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only
103780 support files up to 8 gigabytes in size.

103781 **Issue 6**

103782 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

103783 Support has been added for symbolic links in the options and interchange formats.

103784 A new format has been devised, based on extensions to **ustar**.

103785 References to the “extended” **tar** and **cpio** formats derived from the POSIX.1-1990
103786 standard have been changed to remove the “extended” adjective because this could cause
103787 confusion with the extended **tar** header added in this version. (All references to **tar** are
103788 actually to **ustar**.)

103789 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

103790 IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir()* and *mkfifo()* calls can
103791 ignore an [EEXIST] error when extracting an archive.

103792 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when
103793 in **read** mode.

103794 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option.

103795 IEEE PASC Interpretation 1003.2 #195 is applied.

103796 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**,
103797 and **-I** options.

103798 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of
103799 the *pax* process into certain fields. This change provides a method for the implementation to
103800 ensure that different instances of *pax* extracting a file named **/a/b/foo** will not collide when
103801 processing the extended header information associated with **foo**.

103802 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing **-x B** to **-x pax** in
103803 the OPTIONS section.

103804 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to
103805 be consistent with the normative text.

103806 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/21 is applied, updating the
103807 DESCRIPTION to describe the behavior when files to be linked are symbolic links and the
103808 system is not capable of making hard links to symbolic links.

- 103809 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/22 is applied, updating the OPTIONS
103810 section to describe the behavior for how multiple **-odelete=pattern** options are to be handled.
- 103811 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/23 is applied, updating the **write** option
103812 within the OPTIONS section.
- 103813 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/24 is applied, adding a paragraph into
103814 the OPTIONS section that states that specifying more than one of the mutually-exclusive options
103815 (**-H** and **-L**) is not considered an error and that the last option specified will determine the
103816 behavior of the utility.
- 103817 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/25 is applied, removing the *ctime*
103818 paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of
103819 the *ctime* keyword for the *pax* extended header, in that the *st_ctime* member of the **stat** structure
103820 does not refer to a file creation time. No field in the standard **stat** structure from **<sys/stat.h>**
103821 includes a file creation time.
- 103822 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/26 is applied, making it clear that *typeflag*
103823 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that
103824 are aliased via symbolic links.
- 103825 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/27 is applied, clarifying the *cpio c_nlink*
103826 field.
- 103827 **Issue 7**
- 103828 Austin Group Interpretations 1003.1-2001 #011, #036, #086, and #109 are applied.
- 103829 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
103830 *LC_MESSAGES* environment variable.
- 103831 SD5-XCU-ERN-2 is applied, making **-c** and **-n** mutually-exclusive in the SYNOPSIS.
- 103832 SD5-XCU-ERN-3 is applied, revising the default behavior of **-H** and **-L**.
- 103833 SD5-XCU-ERN-5, SD5-XCU-ERN-6, SD5-XCU-ERN-7, SD5-XCU-ERN-60 are applied.
- 103834 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 103835 The *pax* utility is no longer allowed to create separate identical symbolic links when extracting
103836 linked symbolic links from an archive.
- 103837 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0128 [260], XCU/TC1-2008/0129
103838 [261], XCU/TC1-2008/0130 [261], XCU/TC1-2008/0131 [313], and XCU/TC1-2008/0132 [233]
103839 are applied.
- 103840 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0152 [886], XCU/TC2-2008/0153
103841 [814], XCU/TC2-2008/0154 [886], and XCU/TC2-2008/0155 [707] are applied.

103842 **NAME**103843 pr *pr* 'print files103844 **SYNOPSIS**

103845 pr [+*page*] [-*column*] [-adFmrt] [-e[*char*][*gap*]] [-h *header*] [-i[*char*][*gap*]]
 103846 XSI [-l *lines*] [-n[*char*][*width*]] [-o *offset*] [-s[*char*]] [-w *width*] [-fp]
 103847 [*file...*]

103848 **DESCRIPTION**

103849 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be
 103850 read, formatted, and written to standard output. By default, the input shall be separated into
 103851 66-line pages, each with:

103852 A 5-line header that includes the page number, date, time, and the pathname of the file

103853 A 5-line trailer consisting of blank lines

103854 If standard output is associated with a terminal, diagnostic messages shall be deferred until the
 103855 *pr* utility has completed processing.

103856 When options specifying multi-column output are specified, output text columns shall be of
 103857 equal width; input lines that do not fit into a text column shall be truncated. By default, text
 103858 columns shall be separated with at least one <blank>.

103859 **OPTIONS**

103860 The *pr* utility shall conform to XBD [Section 12.2](#) (on page 216), except that: the *page* option has a
 103861 '+' delimiter; *page* and *column* can be multi-digit numbers; some of the option-arguments are
 103862 optional; and some of the option-arguments cannot be specified as separate arguments from the
 103863 preceding option letter. In particular, the *-s* option does not allow the option letter to be
 103864 separated from its argument, and the options *-e*, *-i*, and *-n* require that both arguments, if
 103865 present, not be separated from the option letter.

103866 The following options shall be supported. In the following option descriptions, *column*, *lines*,
 103867 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

103868 *+page* Begin output at page number *page* of the formatted input.

103869 *-column* Produce multi-column output that is arranged in *column* columns (the default shall
 103870 be 1) and is written down each column in the order in which the text is received
 103871 from the input file. This option should not be used with *-m*. The options *-e* and *-i*
 103872 shall be assumed for multiple text-column output. Whether or not text columns are
 103873 produced with identical vertical lengths is unspecified, but a text column shall
 103874 never exceed the length of the page (see the *-l* option). When used with *-t*, use the
 103875 minimum number of lines to write the output.

103876 *-a* Modify the effect of the *-column* option so that the columns are filled across the
 103877 page in a round-robin order (for example, when *column* is 2, the first input line
 103878 heads column 1, the second heads column 2, the third is the second line in column
 103879 1, and so on).

103880 *-d* Produce output that is double-spaced; append an extra <newline> following every
 103881 <newline> found in the input.

103882 *-e[*char*][*gap*]*

103883 Expand each input <tab> to the next greater column position specified by the
 103884 formula $n * gap + 1$, where *n* is an integer > 0. If *gap* is zero or is omitted, it shall
 103885 default to 8. All <tab> characters in the input shall be expanded into the
 103886 appropriate number of <space> characters. If any non-digit character, *char*, is
 103887 specified, it shall be used as the input <tab>. If the first character of the *-e* option-

- 103888 argument is a digit, the entire option-argument shall be assumed to be *gap*.
- 103889 XSI **-f** Use a <form-feed> for new pages, instead of the default behavior that uses a
103890 sequence of <newline> characters. Pause before beginning the first page if the
103891 standard output is associated with a terminal.
- 103892 **-F** Use a <form-feed> for new pages, instead of the default behavior that uses a
103893 sequence of <newline> characters.
- 103894 **-h header** Use the string *header* to replace the contents of the *file* operand in the page header.
- 103895 **-i[*char*][*gap*]** In output, replace <space> characters with <tab> characters wherever one or more
103896 adjacent <space> characters reach column positions $gap+1$, $2 * gap+1$, $3 * gap+1$, and
103897 so on. If *gap* is zero or is omitted, default tab settings at every eighth column
103898 position shall be assumed. If any non-digit character, *char*, is specified, it shall be
103899 used as the output <tab>. If the first character of the **-i** option-argument is a digit,
103900 the entire option-argument shall be assumed to be *gap*.
- 103901 **-l lines** Override the 66-line default and reset the page length to *lines*. If *lines* is not greater
103902 than the sum of both the header and trailer depths (in lines), the *pr* utility shall
103903 suppress both the header and trailer, as if the **-t** option were in effect.
- 103904 **-m** Merge files. Standard output shall be formatted so the *pr* utility writes one line
103905 from each file specified by a *file* operand, side by side into text columns of equal
103906 fixed widths, in terms of the number of column positions. Implementations shall
103907 support merging of at least nine *file* operands.
- 103908 **-n[*char*][*width*]**
103909 Provide *width*-digit line numbering (default for *width* shall be 5). The number shall
103910 occupy the first *width* column positions of each text column of default output or
103911 each line of **-m** output. If *char* (any non-digit character) is given, it shall be
103912 appended to the line number to separate it from whatever follows (default for *char*
103913 is a <tab>).
- 103914 **-o offset** Each line of output shall be preceded by offset <space> characters. If the **-o** option
103915 is not specified, the default offset shall be zero. The space taken is in addition to the
103916 output line width (see the **-w** option below).
- 103917 **-p** Pause before beginning each page if the standard output is directed to a terminal
103918 (*pr* shall write an <alert> to standard error and wait for a <carriage-return> to be
103919 read on */dev/tty*).
- 103920 **-r** Write no diagnostic reports on failure to open files.
- 103921 **-s[*char*]** Separate text columns by the single character *char* instead of by the appropriate
103922 number of <space> characters (default for *char* shall be <tab>).
- 103923 **-t** Write neither the five-line identifying header nor the five-line trailer usually
103924 supplied for each page. Quit writing after the last line of each file without spacing
103925 to the end of the page.
- 103926 **-w width** Set the width of the line to *width* column positions for multiple text-column output
103927 only. If the **-w** option is not specified and the **-s** option is not specified, the default
103928 width shall be 72. If the **-w** option is not specified and the **-s** option is specified,
103929 the default width shall be 512.
- 103930 For single column output, input lines shall not be truncated.

103931 **OPERANDS**

103932 The following operand shall be supported:

103933 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*
103934 operand is '-', the standard input shall be used.

103935 **STDIN**

103936 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
103937 See the INPUT FILES section.

103938 **INPUT FILES**

103939 The input files shall be text files.

103940 The file `/dev/tty` shall be used to read responses required by the `-p` option.

103941 **ENVIRONMENT VARIABLES**

103942 The following environment variables shall affect the execution of *pr*:

103943 *LANG* Provide a default value for the internationalization variables that are unset or null.
103944 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
103945 used to determine the values of locale categories.)

103946 *LC_ALL* If set to a non-empty string value, override the values of all the other
103947 internationalization variables.

103948 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
103949 characters (for example, single-byte as opposed to multi-byte characters in
103950 arguments and input files) and which characters are defined as printable (character
103951 class **print**). Non-printable characters are still written to standard output, but are
103952 not counted for the purpose for column-width and line-length calculations.

103953 *LC_MESSAGES*

103954 Determine the locale that should be used to affect the format and contents of
103955 diagnostic messages written to standard error.

103956 *LC_TIME* Determine the format of the date and time for use in writing header lines.

103957 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

103958 *TZ* Determine the timezone used to calculate date and time strings written in header
103959 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

103960 **ASYNCHRONOUS EVENTS**

103961 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error
103962 messages to the screen before terminating.

103963 **STDOUT**

103964 The *pr* utility output shall be a paginated version of the original file (or files). This pagination
103965 shall be accomplished using either <form-feed> characters or a sequence of <newline>
103966 XSI characters, as controlled by the `-F` or `-f` option. Page headers shall be generated unless the `-t`
103967 option is specified. The page headers shall be of the form:

103968 `"\n\n%s %s Page %d\n\n", <output of date>, <file>, <page number>`

103969 In the POSIX locale, the <output of date> field, representing the date and time of last modification
103970 of the input file (or the current date and time if the input file is standard input), shall be
103971 equivalent to the output of the following command as it would appear if executed at the given
103972 time:

103973 `date "+%b %e %H:%M %Y"`

103974 without the trailing <newline>, if the page being written is from standard input. If the page
 103975 being written is not from standard input, in the POSIX locale, the same format shall be used, but
 103976 the time used shall be the modification time of the file corresponding to *file* instead of the current
 103977 time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format and
 103978 order of presentation of this field may be used.

103979 If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null
 103980 string.

103981 If the **-h** option is specified, the <*file*> field shall be replaced by the *header* argument.

103982 **STDERR**

103983 The standard error shall be used for diagnostic messages and for alerting the terminal when **-p**
 103984 is specified.

103985 **OUTPUT FILES**

103986 None.

103987 **EXTENDED DESCRIPTION**

103988 None.

103989 **EXIT STATUS**

103990 The following exit values shall be returned:

103991 0 Successful completion.

103992 >0 An error occurred.

103993 **CONSEQUENCES OF ERRORS**

103994 Default.

103995 **APPLICATION USAGE**

103996 A conforming application must protect its first operand, if it starts with a <plus-sign>, by
 103997 preceding it with the "--" argument that denotes the end of the options. For example, *pr+x*
 103998 could be interpreted as an invalid page number or a *file* operand.

103999 **EXAMPLES**

104000 1. Print a numbered list of all files in the current directory:

```
104001 ls -a | pr -n -h "Files in $(pwd)."
```

104002 2. Print **file1** and **file2** as a double-spaced, three-column listing headed by ``file list``:

```
104003 pr -3d -h "file list" file1 file2
```

104004 3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:

```
104005 pr -e9 -t <file1 >file2
```

104006 **RATIONALE**

104007 This utility is one of those that does not follow the Utility Syntax Guidelines because of its
 104008 historical origins. The standard developers could have added new options that obeyed the
 104009 guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are
 104010 examples of both actions in this volume of POSIX.1-2017. Because of its widespread use by
 104011 historical applications, the standard developers decided to exempt this version of *pr* from many
 104012 of the guidelines.

104013 Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options
 104014 whether presented as part of the same argument or as a separate argument to *pr*, as suggested by
 104015 the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical

104016 practice because they are frequently specified without their optional arguments. If a <blank>
104017 were allowed before the option-argument in these cases, a *file* operand could mistakenly be
104018 interpreted as an option-argument in historical applications.

104019 The text about the minimum number of lines in multi-column output was included to ensure
104020 that a best effort is made in balancing the length of the columns. There are known historical
104021 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines
104022 and a second of 4. Although this is not a problem when a full page with headers and trailers is
104023 produced, it would be relatively useless when used with *-t*.

104024 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.
104025 BSD uses it as described here for the *-F* option; System V uses it to change trailing <newline>
104026 characters on each page to a <form-feed> and, if standard output is a TTY device, sends an
104027 <alert> to standard error and reads a line from */dev/tty* before the first page. There were strong
104028 arguments from both sides of this issue concerning historical practice and as a result the *-F*
104029 option was added. XSI-conformant systems support the System V historical actions for the *-f*
104030 option.

104031 The <output of date> field in the *-l* format is specified only for the POSIX locale. As noted, the
104032 format can be different in other locales. No mechanism for defining this is present in this volume
104033 of POSIX.1-2017, as the appropriate vehicle is a message catalog; that is, the format should be
104034 specified as a ``message''.

104035 **FUTURE DIRECTIONS**

104036 None.

104037 **SEE ALSO**

104038 *expand*, *lp*

104039 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

104040 **CHANGE HISTORY**

104041 First released in Issue 2.

104042 **Issue 6**

104043 The following new requirements on POSIX implementations derive from alignment with the
104044 Single UNIX Specification:

104045 The *-p* option is added.

104046 The normative text is reworded to avoid use of the term ``must'' for application requirements.

104047 **Issue 7**

104048 PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied.

104049 Austin Group Interpretation 1003.1-2001 #093 is applied.

104050 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104051 **NAME**

104052 printf ‡write formatted output

104053 **SYNOPSIS**104054 printf *format* [*argument...*]104055 **DESCRIPTION**104056 The *printf* utility shall write formatted operands to the standard output. The *argument* operands
104057 shall be formatted under control of the *format* operand.104058 **OPTIONS**

104059 None.

104060 **OPERANDS**

104061 The following operands shall be supported:

104062 *format* A string describing the format to use to write the remaining operands. See the
104063 EXTENDED DESCRIPTION section.104064 *argument* The strings to be written to standard output, under the control of *format*. See the
104065 EXTENDED DESCRIPTION section.104066 **STDIN**

104067 Not used.

104068 **INPUT FILES**

104069 None.

104070 **ENVIRONMENT VARIABLES**104071 The following environment variables shall affect the execution of *printf*:104072 *LANG* Provide a default value for the internationalization variables that are unset or null.
104073 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
104074 used to determine the values of locale categories.)104075 *LC_ALL* If set to a non-empty string value, override the values of all the other
104076 internationalization variables.104077 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
104078 characters (for example, single-byte as opposed to multi-byte characters in
104079 arguments).104080 *LC_MESSAGES*104081 Determine the locale that should be used to affect the format and contents of
104082 diagnostic messages written to standard error.104083 *LC_NUMERIC*104084 Determine the locale for numeric formatting. It shall affect the format of numbers
104085 written using the e, E, f, g, and G conversion specifier characters (if supported).104086 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.104087 **ASYNCHRONOUS EVENTS**

104088 Default.

104089 **STDOUT**

104090 See the EXTENDED DESCRIPTION section.

104091 **STDERR**

104092 The standard error shall be used only for diagnostic messages.

104093 **OUTPUT FILES**

104094 None.

104095 **EXTENDED DESCRIPTION**

104096 The *format* operand shall be used as the *format* string described in XBD Chapter 5 (on page 121)
104097 with the following exceptions:

- 104098 1. A <space> in the format string, in any context other than a flag of a conversion
104099 specification, shall be treated as an ordinary character that is copied to the output.
- 104100 2. A '\Δ' character in the format string shall be treated as a '\Δ' character, not as a <space>.
- 104101 3. In addition to the escape sequences shown in XBD Chapter 5 (on page 121) ('\ ', '\a',
104102 '\b', '\f', '\n', '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit
104103 octal number, shall be written as a byte with the numeric value specified by the octal
104104 number.
- 104105 4. The implementation shall not precede or follow output from the *d* or *u* conversion
104106 specifiers with <blank> characters not specified by the *format* operand.
- 104107 5. The implementation shall not precede output from the *o* conversion specifier with zeros
104108 not specified by the *format* operand.
- 104109 6. The *a*, *A*, *e*, *E*, *f*, *F*, *g*, and *G* conversion specifiers need not be supported.
- 104110 7. An additional conversion specifier character, *b*, shall be supported as follows. The
104111 argument shall be taken to be a string that can contain <backslash>-escape sequences.
104112 The following <backslash>-escape sequences shall be supported:
 - 104113 ‡\kēEscape sequences listed in XBD Chapter 5 (on page 121) ('\ ', '\a', '\b',
104114 '\f', '\n', '\r', '\t', '\v'), which shall be converted to the characters they
104115 represent
 - 104116 ‡"\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be
104117 converted to a byte with the numeric value specified by the octal number
 - 104118 ‡'\c', which shall not be written and shall cause *printf* to ignore any remaining
104119 characters in the string operand containing it, any remaining string operands, and
104120 any additional characters in the *format* operand
- 104121 The interpretation of a <backslash> followed by any other sequence of characters is
104122 unspecified.
- 104123 Bytes from the converted string shall be written until the end of the string or the number
104124 of bytes indicated by the precision specification is reached. If the precision is omitted, it
104125 shall be taken to be infinite, so all bytes up to the end of the converted string shall be
104126 written.
- 104127 8. For each conversion specification that consumes an argument, the next *argument* operand
104128 shall be evaluated and converted to the appropriate type for the conversion as specified
104129 below.
- 104130 9. The *format* operand shall be reused as often as necessary to satisfy the *argument* operands.
104131 Any extra *b*, *c*, or *s* conversion specifiers shall be evaluated as if a null string argument
104132 were supplied; other extra conversion specifications shall be evaluated as if a zero
104133 argument were supplied. If the *format* operand contains no conversion specifications and
104134 *argument* operands are present, the results are unspecified.

- 104135 10. If a character sequence in the *format* operand begins with a '%' character, but does not
104136 form a valid conversion specification, the behavior is unspecified.
- 104137 11. The argument to the *c* conversion specifier can be a string containing zero or more bytes.
104138 If it contains one or more bytes, the first byte shall be written and any additional bytes
104139 shall be ignored. If the argument is an empty string, it is unspecified whether nothing is
104140 written or a null byte is written.

104141 The *argument* operands shall be treated as strings if the corresponding conversion specifier is *b*,
104142 *c*, or *s*, and shall be evaluated as if by the *strtod()* function if the corresponding conversion
104143 specifier is *a*, *A*, *e*, *E*, *f*, *F*, *g*, or *G*. Otherwise, they shall be evaluated as unsuffixed C integer
104144 constants, as described by the ISO C standard, with the following extensions:

104145 A leading <plus-sign> or <hyphen-minus> shall be allowed.

104146 If the leading character is a single-quote or double-quote, the value shall be the numeric
104147 value in the underlying codeset of the character following the single-quote or double-
104148 quote.

104149 Suffixed integer constants may be allowed.

104150 If an *argument* operand cannot be completely converted into an internal value appropriate to the
104151 corresponding conversion specification, a diagnostic message shall be written to standard error
104152 and the utility shall not exit with a zero exit status, but shall continue processing any remaining
104153 operands and shall write the value accumulated at the time the error was detected to standard
104154 output.

104155 It shall not be considered an error if an *argument* operand is not completely used for a *b*, *c*, or *s*
104156 conversion.

104157 EXIT STATUS

104158 The following exit values shall be returned:

104159 0 Successful completion.

104160 >0 An error occurred.

104161 CONSEQUENCES OF ERRORS

104162 Default.

104163 APPLICATION USAGE

104164 The floating-point formatting conversion specifications of *printf()* are not required because all
104165 arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations
104166 and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-
104167 point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility
104168 cannot really be used to format *bc* output; it does not support arbitrary precision.)
104169 Implementations are encouraged to support the floating-point conversions as an extension.

104170 Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of
104171 POSIX.1-2017 on which it is based, makes no special provision for dealing with multi-byte
104172 characters when using the *%c* conversion specification or when a precision is specified in a *%b* or
104173 *%s* conversion specification. Applications should be extremely cautious using either of these
104174 features when there are multi-byte characters in the character set.

104175 No provision is made in this volume of POSIX.1-2017 which allows field widths and precisions
104176 to be specified as '*' since the '*' can be replaced directly in the *format* operand using shell
104177 variable substitution. Implementations can also provide this feature as an extension if they so
104178 choose.

104179 Hexadecimal character constants as defined in the ISO C standard are not recognized in the
 104180 *format* operand because there is no consistent way to detect the end of the constant. Octal
 104181 character constants are limited to, at most, three octal digits, but hexadecimal character constants
 104182 are only terminated by a non-hex-digit character. In the ISO C standard, the "##" concatenation
 104183 operator can be used to terminate a constant and follow it with a hexadecimal character to be
 104184 written. In the shell, concatenation occurs before the *printf* utility has a chance to parse the end
 104185 of the hexadecimal constant.

104186 The %b conversion specification is not part of the ISO C standard; it has been added here as a
 104187 portable way to process <backslash>-escapes expanded in string operands as provided by the
 104188 *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 2674) for ways to use
 104189 *printf* as a replacement for all of the traditional versions of the *echo* utility.

104190 If an argument cannot be parsed correctly for the corresponding conversion specification, the
 104191 *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end
 104192 of an argument being used for a numeric conversion shall be reported as errors.

104193 EXAMPLES

104194 To alert the user and then print and read a series of prompts:

```
104195 printf "\aPlease fill in the following: \nName: "  
104196 read name  
104197 printf "Phone number: "  
104198 read phone
```

104199 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and
 104200 print them out. The numbers are right-justified and separated by a single <tab>. The percentage
 104201 is written to one decimal place of accuracy:

```
104202 while read right wrong ; do  
104203     percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)  
104204     printf "%2d right\t%2d wrong\t(%s%%)\n" \  
104205         $right $wrong $percent  
104206 done < database_file
```

104207 The command:

```
104208 printf "%5d%4d\n" 1 21 321 4321 54321
```

104209 produces:

```
104210     1  21  
104211     3214321  
104212 54321  0
```

104213 Note that the *format* operand is used three times to print all of the given strings and that a '0'
 104214 was supplied by *printf* to satisfy the last %4d conversion specification.

104215 The *printf* utility is required to notify the user when conversion errors are detected while
 104216 producing numeric output; thus, the following results would be expected on an implementation
 104217 with 32-bit twos-complement integers when %d is specified as the *format* operand:

	Argument	Standard Output	Diagnostic Output
104218	5a	5	printf: "5a" not completely converted
104219	9999999999	2147483647	printf: "9999999999" arithmetic overflow
104220	-9999999999	-2147483648	printf: "-9999999999" arithmetic overflow
104221	ABC	0	printf: "ABC" expected numeric value

104222 The diagnostic message format is not specified, but these examples convey the type of
 104223 information that should be reported. Note that the value shown on standard output is what
 104224 would be expected as the return value from the *strtol()* function as defined in the System
 104225 Interfaces volume of POSIX.1-2017. A similar correspondence exists between *%u* and *strtoul()*
 104226 and *%e*, *%f*, and *%g* (if the implementation supports floating-point conversions) and *strtod()*.

104227 In a locale using the ISO/IEC 646: 1991 standard as the underlying codeset, the command:

```
104228 printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"
```

104229 produces:

104230 3 Numeric value of constant 3

104231 3 Numeric value of constant 3

104232 -3 Numeric value of constant -3

104233 51 Numeric value of the character '3' in the ISO/IEC 646: 1991 standard codeset

104234 43 Numeric value of the character '+' in the ISO/IEC 646: 1991 standard codeset

104235 45 Numeric value of the character '-' in the ISO/IEC 646: 1991 standard codeset

104236 Note that in a locale with multi-byte characters, the value of a character is intended to be the
 104237 value of the equivalent of the *wchar_t* representation of the character as described in the System
 104238 Interfaces volume of POSIX.1-2017.

104239 RATIONALE

104240 The *printf* utility was added to provide functionality that has historically been provided by *echo*.
 104241 However, due to irreconcilable differences in the various versions of *echo* extant, the version has
 104242 few special features, leaving those to this new *printf* utility, which is based on one in the Ninth
 104243 Edition system.

104244 The EXTENDED DESCRIPTION section almost exactly matches the *printf()* function in the
 104245 ISO C standard, although it is described in terms of the file format notation in XBD [Chapter 5](#)
 104246 (on page 121).

104247 Earlier versions of this standard specified that arguments for all conversions other than *b*, *c*, and
 104248 *s* were evaluated in the same way (as C constants, but with stated exceptions). For
 104249 implementations supporting the floating-point conversions it was not clear whether integer
 104250 conversions need only accept integer constants and floating-point conversions need only accept
 104251 floating-point constants, or whether both types of conversions should accept both types of
 104252 constants. Also by not distinguishing between them, the requirement relating to a leading
 104253 single-quote or double-quote applied to floating-point conversions even though this provided
 104254 no useful functionality to applications that was not already available through the integer
 104255 conversions. The current standard clarifies the situation by specifying that the arguments for
 104256 floating-point conversions are evaluated as if by *strtod()*, and the arguments for integer
 104257 conversions are evaluated as C integer constants, with the special treatment of leading single-
 104258 quote and double-quote applying only to integer conversions.

104261 **FUTURE DIRECTIONS**

104262 None.

104263 **SEE ALSO**104264 *awk, bc, echo*104265 XBD [Chapter 5](#) (on page 121), [Chapter 8](#) (on page 173)104266 XSH *fprintf(), strtod()*104267 **CHANGE HISTORY**

104268 First released in Issue 4.

104269 **Issue 7**

104270 Austin Group Interpretations 1003.1-2001 #175 and #177 are applied.

104271 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104272 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0156 [727], XCU/TC2-2008/0157
104273 [727,932], XCU/TC2-2008/0158 [584], and XCU/TC2-2008/0159 [727] are applied.

104274 **NAME**104275 prs ‡'print an SCCS file **DEVELOPMENT**)104276 **SYNOPSIS**104277 XSI prs [-a] [-d *dataspec*] [-r[*SID*]] *file...*104278 prs [-e|-l] -c *cutoff* [-d *dataspec*] *file...*104279 prs [-e|-l] -r[*SID*] [-d *dataspec*] *file...*104280 **DESCRIPTION**104281 The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied
104282 format.104283 **OPTIONS**104284 The *prs* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the `-r` option has an
104285 optional option-argument. This optional option-argument cannot be presented as a separate
104286 argument. The following options shall be supported:104287 `-d dataspec` Specify the output data specification. The *dataspec* shall be a string consisting of
104288 SCCS file *data keywords* (see [Data Keywords](#), on page 3119) interspersed with
104289 optional user-supplied text.104290 `-r[SID]` Specify the SCCS identification string (SID) of a delta for which information is
104291 desired. If no *SID* option-argument is specified, the SID of the most recently
104292 created delta shall be assumed.104293 `-e` Request information for all deltas created earlier than and including the delta
104294 designated via the `-r` option or the date-time given by the `-c` option.104295 `-l` Request information for all deltas created later than and including the delta
104296 designated via the `-r` option or the date-time given by the `-c` option.104297 `-c cutoff` Indicate the *cutoff* date-time, in the form:104298 `YY[MM[DD[HH[MM[SS]]]]]`104299 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999
104300 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.104301 **Note:** It is expected that in a future version of this standard the default century inferred
104302 from a 2-digit year will change. (This would apply to all commands accepting a
104303 2-digit year as input.)104304 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
104305 date-time shall be included in the output. Units omitted from the date-time default
104306 to their maximum possible values; for example, `-c 7502` is equivalent to
104307 `-c 750228235959`.104308 `-a` Request writing of information for both removed ‡that is, *delta type=R* (see
104309 *rm~~del~~*) ‡and existing ‡that is, *delta type=D*, ‡deltas. If the `-a` option is not
104310 specified, information for existing deltas only shall be provided.104311 **OPERANDS**

104312 The following operand shall be supported:

104313 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*
104314 utility shall behave as though each file in the directory were specified as a named
104315 file, except that non-SCCS files (last component of the pathname does not begin
104316 with *s*.) and unreadable files shall be silently ignored.

104317 If exactly one *file* operand appears, and it is '-', the standard input shall be read;
 104318 each line of the standard input shall be taken to be the name of an SCCS file to be
 104319 processed. Non-SCCS files and unreadable files shall be silently ignored.

104320 STDIN

104321 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each
 104322 line of the text file shall be interpreted as an SCCS pathname.

104323 INPUT FILES

104324 Any SCCS files displayed are files of an unspecified format.

104325 ENVIRONMENT VARIABLES

104326 The following environment variables shall affect the execution of *prs*:

104327 *LANG* Provide a default value for the internationalization variables that are unset or null.
 104328 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
 104329 used to determine the values of locale categories.)

104330 *LC_ALL* If set to a non-empty string value, override the values of all the other
 104331 internationalization variables.

104332 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 104333 characters (for example, single-byte as opposed to multi-byte characters in
 104334 arguments and input files).

104335 *LC_MESSAGES*

104336 Determine the locale that should be used to affect the format and contents of
 104337 diagnostic messages written to standard error.

104338 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

104339 ASYNCHRONOUS EVENTS

104340 Default.

104341 STDOUT

104342 The standard output shall be a text file whose format is dependent on the data keywords
 104343 specified with the *-d* option.

104344 Data Keywords

104345 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an
 104346 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple
 104347 times.

104348 The information written by *prs* shall consist of:

- 104349 1. The user-supplied text
- 104350 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data
 104351 keywords in the order of appearance in the *dataspec*

104352 The format of a data keyword value shall either be simple ('S'), in which keyword substitution
 104353 is direct, or multi-line ('M').

104354 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be
 104355 specified by '\t' and <newline> by '\n'. When the *-r* option is not specified, the default
 104356 *dataspec* shall be:

104357 :PN::\n\n

104358 and the following *dataspec* shall be used for each selected delta:

104359 :Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:

SCCS File Data Keywords					
Keyword	Data Item	File Section	Value	Format	
104360					
104361					
104362	:Dt:	Delta information	Delta Table	See below*	S
104363	:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
104364	:Li:	Lines inserted by Delta	"	nnnnn***	S
104365	:Ld:	Lines deleted by Delta	"	nnnnn***	S
104366	:Lu:	Lines unchanged by Delta	"	nnnnn***	S
104367	:DT:	Delta type	"	D or R	S
104368	:I:	SCCS ID string (SID)	"	See below**	S
104369	:R:	Release number	"	nnnn	S
104370	:L:	Level number	"	nnnn	S
104371	:B:	Branch number	"	nnnn	S
104372	:S:	Sequence number	"	nnnn	S
104373	:D:	Date delta created	"	:Dy:/:Dm:/:Dd:	S
104374	:Dy:	Year delta created	"	nn	S
104375	:Dm:	Month delta created	"	nn	S
104376	:Dd:	Day delta created	"	nn	S
104377	:T:	Time delta created	"	:Th:::Tm:::Ts:	S
104378	:Th:	Hour delta created	"	nn	S
104379	:Tm:	Minutes delta created	"	nn	S
104380	:Ts:	Seconds delta created	"	nn	S
104381	:P:	Programmer who created Delta	"	logname	S
104382	:DS:	Delta sequence number	"	nnnn	S
104383	:DP:	Predecessor Delta sequence number	"	nnnn	S
104384					
104385	:DI:	Sequence number of deltas included, excluded, or ignored	"	:Dn:/:Dx:/:Dg:	S
104386					
104387	:Dn:	Deltas included (sequence #)	"	:DS: :DS: ...	S
104388	:Dx:	Deltas excluded (sequence #)	"	:DS: :DS: ...	S
104389	:Dg:	Deltas ignored (sequence #)	"	:DS: :DS: ...	S
104390	:MR:	MR numbers for delta	"	text	M
104391	:C:	Comments for delta	"	text	M
104392	:UN:	User names	User Names	text	M
104393	:FL:	Flag list	Flags	text	M
104394	:Y:	Module type flag	"	text	S
104395	:MF:	MR validation flag	"	yes or no	S
104396	:MP:	MR validation program name	"	text	S
104397	:KF:	Keyword error, warning flag	"	yes or no	S
104398	:KV:	Keyword validation string	"	text	S
104399	:BF:	Branch flag	"	yes or no	S
104400	:J:	Joint edit flag	"	yes or no	S
104401	:LK:	Locked releases	"	:R: ...	S
104402	:Q:	User-defined keyword	"	text	S
104403	:M:	Module name	"	text	S
104404	:FB:	Floor boundary	"	:R:	S
104405	:CB:	Ceiling boundary	"	:R:	S
104406	:Ds:	Default SID	"	:I:	S
104407	:ND:	Null delta flag	"	yes or no	S

SCCS File Data Keywords				
Keyword	Data Item	File Section	Value	Format
104408				
104409				
104410	:FD:	File descriptive text	Comments	text M
104411	:BD:	Body	Body	text M
104412	:GB:	Gotten body	"	text M
104413	:W:	A form of <i>what</i> string	N/A	:Z::M:\t:I: S
104414	:A:	A form of <i>what</i> string	N/A	:Z::Y: :M: :I::Z: S
104415	:Z:	<i>what</i> string delimiter	N/A	@ (#) S
104416	:F:	SCCS filename	N/A	text S
104417	:PN:	SCCS file pathname	N/A	text S
104418	*	:Dt:=:DT: :I: :D: :T: :P: :DS: :DP:		
104419	**	:R::L::B::S: if the delta is a branch delta (:BF:= =yes)		
104420		:R::L: if the delta is not a branch delta (:BF:= =no)		
104421	***	The line statistics are capped at 99 999. For example, if 100 000 lines were unchanged in a		
104422		certain revision, :Lu: shall produce the value 99 999.		
104423	STDERR			
104424		The standard error shall be used only for diagnostic messages.		
104425	OUTPUT FILES			
104426		None.		
104427	EXTENDED DESCRIPTION			
104428		None.		
104429	EXIT STATUS			
104430		The following exit values shall be returned:		
104431		0 Successful completion.		
104432		>0 An error occurred.		
104433	CONSEQUENCES OF ERRORS			
104434		Default.		
104435	APPLICATION USAGE			
104436		None.		
104437	EXAMPLES			
104438		1. The following example:		
104439		<code>prs -d "User Names for :F: are:\n:UN:" s.file</code>		
104440		might write to standard output:		
104441		User Names for s.file are:		
104442		xyz		
104443		131		
104444		abc		
104445		2. The following example:		
104446		<code>prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file</code>		
104447		might write to standard output:		
104448		Delta for pgm main.c: 3.7 - 77/12/01 By cas		

104449 3. As a special case:
 104450 prs s.file
 104451 might write to standard output:
 104452 s.file:
 104453 <blank line>
 104454 D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000
 104455 MRs:
 104456 b178-12345
 104457 b179-54321
 104458 COMMENTS:
 104459 this is the comment line for s.file initial delta
 104460 <blank line>

104461 for each delta table entry of the **D** type. The only option allowed to be used with this
 104462 special case is the **-a** option.

104463 **RATIONALE**
 104464 None.

104465 **FUTURE DIRECTIONS**
 104466 None.

104467 **SEE ALSO**
 104468 *admin, delta, get, what*
 104469 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

104470 **CHANGE HISTORY**
 104471 First released in Issue 2.

104472 **Issue 5**
 104473 The phrase “in which keyword substitution is followed by a <newline>” is deleted from the end
 104474 of the second paragraph of [Data Keywords](#) (on page 3119).
 104475 The interpretation of the *YY* component of the **-c cutoff** argument is noted.

104476 **Issue 6**
 104477 The normative text is reworded to emphasize the term “shall” for implementation requirements.
 104478 The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a
 104479 note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.
 104480 The Open Group Interpretation PIN4C.00009 is applied.

104481 **Issue 7**
 104482 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104483 **NAME**

104484 ps — report process status

104485 **SYNOPSIS**

```
104486 XSI ps [-aA] [-defl] [-g grouplist] [-G grouplist]
104487 [-n namelist] [-o format]... [-p proclist] [-t termlist]
104488 [-u userlist] [-U userlist]
```

104489 **DESCRIPTION**

104490 The *ps* utility shall write information about processes, subject to having appropriate privileges to
 104491 obtain information about those processes.

104492 By default, *ps* shall select all processes with the same effective user ID as the current user and the
 104493 same controlling terminal as the invoker.

104494 **OPTIONS**

104495 The *ps* utility shall conform to XBD [Section 12.2](#) (on page 216).

104496 The following options shall be supported:

- 104497 **-a** Write information for all processes associated with terminals. Implementations
 104498 may omit session leaders from this list.
- 104499 **-A** Write information for all processes.
- 104500 XSI **-d** Write information for all processes, except session leaders.
- 104501 XSI **-e** Write information for all processes. (Equivalent to **-A**.)
- 104502 XSI **-f** Generate a **full** listing. (See the **STDOUT** section for the contents of a **full** listing.)
- 104503 XSI **-g grouplist** Write information for processes whose session leaders are given in *grouplist*. The
 104504 application shall ensure that the *grouplist* is a single argument in the form of a
 104505 <blank> or <comma>-separated list.
- 104506 **-G grouplist** Write information for processes whose real group ID numbers are given in
 104507 *grouplist*. The application shall ensure that the *grouplist* is a single argument in the
 104508 form of a <blank> or <comma>-separated list.
- 104509 XSI **-l** Generate a **long** listing. (See **STDOUT** for the contents of a **long** listing.)
- 104510 XSI **-n namelist** Specify the name of an alternative system *namelist* file in place of the default. The
 104511 name of the default file and the format of a *namelist* file are unspecified.
- 104512 **-o format** Write information according to the format specification given in *format*. This is
 104513 fully described in the **STDOUT** section. Multiple **-o** options can be specified; the
 104514 format specification shall be interpreted as the <space>-separated concatenation of
 104515 all the *format* option-arguments.
- 104516 **-p proclist** Write information for processes whose process ID numbers are given in *proclist*.
 104517 The application shall ensure that the *proclist* is a single argument in the form of a
 104518 <blank> or <comma>-separated list.
- 104519 **-t termlist** Write information for processes associated with terminals given in *termlist*. The
 104520 application shall ensure that the *termlist* is a single argument in the form of a
 104521 <blank> or <comma>-separated list. Terminal identifiers shall be given in an
 104522 XSI implementation-defined format. On XSI-conformant systems, they shall be given
 104523 in one of two forms: the device's filename (for example, **tty04**) or, if the device's
 104524 filename starts with **tty**, just the identifier following the characters **tty** (for example,
 104525 "04").

- 104526 XSI **-u *userlist*** Write information for processes whose user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or <comma>-separated list. In the listing, the numerical user ID shall be written unless the **-f** option is used, in which case the login name shall be written.
- 104527
- 104528
- 104529
- 104530
- 104531 **-U *userlist*** Write information for processes whose real user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or <comma>-separated list.
- 104532
- 104533
- 104534 XSI With the exception of **-f**, **-l**, **-n *namelist***, and **-o *format***, all of the options shown are used to select processes. If any are specified, the default list shall be ignored and *ps* shall select the processes represented by the inclusive OR of all the selection-criteria options.
- 104535
- 104536
- 104537 **OPERANDS**
- 104538 None.
- 104539 **STDIN**
- 104540 Not used.
- 104541 **INPUT FILES**
- 104542 None.
- 104543 **ENVIRONMENT VARIABLES**
- 104544 The following environment variables shall affect the execution of *ps*:
- 104545 **COLUMNS** Override the system-selected horizontal display line size, used to determine the number of text columns to display. See XBD [Chapter 8](#) (on page 173) for valid values and results when it is unset or null.
- 104546
- 104547
- 104548 **LANG** Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables used to determine the values of locale categories.)
- 104549
- 104550
- 104551 **LC_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.
- 104552
- 104553 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 104554
- 104555
- 104556 **LC_MESSAGES**
- 104557 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
- 104558
- 104559
- 104560 **LC_TIME** Determine the format and contents of the date and time strings displayed.
- 104561 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 104562 **TZ** Determine the timezone used to calculate date and time strings displayed. If **TZ** is unset or null, an unspecified default timezone shall be used.
- 104563
- 104564 **ASYNCHRONOUS EVENTS**
- 104565 Default.

104566 **STDOUT**

104567 When the **-o** option is not specified, the standard output format is unspecified.

104568 XSI On XSI-conformant systems, the output format shall be as follows. The column headings and
 104569 descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields
 104570 are implementation-defined. The letters 'f' and 'l' (below) indicate the option (**full** or **long**)
 104571 that shall cause the corresponding heading to appear; **all** means that the heading always
 104572 appears. Note that these two options determine only what information is provided for a process;
 104573 they do not determine which processes are listed.

104574	F	(l)	Flags (octal and additive) associated with the process.
104575	S	(l)	The state of the process.
104576	UID	(f,l)	The user ID number of the process owner; the login name is printed 104577 under the -f option.
104578	PID	(all)	The process ID of the process; it is possible to kill a process if this 104579 datum is known.
104580	PPID	(f,l)	The process ID of the parent process.
104581	C	(f,l)	Processor utilization for scheduling.
104582	PRI	(l)	The priority of the process; higher numbers mean lower priority.
104583	NI	(l)	Nice value; used in priority computation.
104584	ADDR	(l)	The address of the process.
104585	SZ	(l)	The size in blocks of the core image of the process.
104586	WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the 104587 process is running.
104588	STIME	(f)	Starting time of the process.
104589	TTY	(all)	The controlling terminal for the process.
104590	TIME	(all)	The cumulative execution time for the process.
104591	CMD	(all)	The command name; the full command name and its arguments are 104592 written under the -f option.

104593 A process that has exited and has a parent, but has not yet been waited for by the parent, shall be
 104594 marked **defunct**.

104595 Under the option **-f**, *ps* tries to determine the command name and arguments given when the
 104596 process was created by examining memory or the swap area. Failing this, the command name, as
 104597 it would appear without the option **-f**, is written in square brackets.

104598 The **-o** option allows the output format to be specified under user control.

104599 The application shall ensure that the format specification is a list of names presented as a single
 104600 argument, <blank> or <comma>-separated. Each variable has a default header. The default
 104601 header can be overridden by appending an <equals-sign> and the new text of the header. The
 104602 rest of the characters in the argument shall be used as the header text. The fields specified shall
 104603 be written in the order specified on the command line, and should be arranged in columns in the
 104604 output. The field widths shall be selected by the system to be at least as wide as the header text
 104605 (default or overridden value). If the header text is null, such as **-o user=**, the field width shall be
 104606 at least as wide as the default header text. If all header text fields are null, no header line shall
 104607 be written.

104608 The following names are recognized in the POSIX locale:

104609	ruser	The real user ID of the process. This shall be the textual user ID, if it can be obtained 104610 and the field width permits, or a decimal representation otherwise.
--------	--------------	---

104611	user	The effective user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
104612		
104613	rgroup	The real group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
104614		
104615	group	The effective group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
104616		
104617	pid	The decimal value of the process ID.
104618	ppid	The decimal value of the parent process ID.
104619	pgid	The decimal value of the process group ID.
104620	pcpu	The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of “recently” in this context is unspecified. The CPU time available is determined in an unspecified manner.
104621		
104622		
104623	vsz	The size of the process in (virtual) memory in 1024 byte units as a decimal integer.
104624	nice	The decimal value of the nice value of the process; see <i>nice</i> .
104625	etime	In the POSIX locale, the elapsed time since the process was started, in the form:
104626		[[<i>dd</i> -] <i>hh</i> :] <i>mm</i> : <i>ss</i>
104627		where <i>dd</i> shall represent the number of days, <i>hh</i> the number of hours, <i>mm</i> the number of minutes, and <i>ss</i> the number of seconds. The <i>dd</i> field shall be a decimal integer. The <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be two-digit decimal integers padded on the left with zeros.
104628		
104629		
104630	time	In the POSIX locale, the cumulative CPU time of the process in the form:
104631		[[<i>dd</i> -] <i>hh</i> : <i>mm</i> : <i>ss</i>
104632		The <i>dd</i> , <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be as described in the etime specifier.
104633	tty	The name of the controlling terminal of the process (if any) in the same format used by the <i>who</i> utility.
104634		
104635	comm	The name of the command being executed (<i>argv</i> [0] value) as a string.
104636	args	The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation-defined whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of <i>ps</i> .
104637		
104638		
104639		
104640		
104641		
104642		
104643		Any field need not be meaningful in all implementations. In such a case a <hyphen-minus>
104644		(' - ') should be output in place of the field value.
104645		Only comm and args shall be allowed to contain <blank> characters; all others shall not. Any implementation-defined variables shall be specified in the system documentation along with the default header and indicating whether the field may contain <blank> characters.
104646		
104647		
104648		The following table specifies the default header to be used in the POSIX locale corresponding to each format specifier.
104649		

104650

Table 4-18 Variable Names and Default Headers in *ps*

104651

Format Specifier	Default Header	Format Specifier	Default Header
args	COMMAND	ppid	PPID
comm	COMMAND	rgroup	RGROUP
etime	ELAPSED	ruser	RUSER
group	GROUP	time	TIME
nice	NI	tty	TT
pcpu	%CPU	user	USER
pgid	PGID	vsz	VSZ
pid	PID		

104652

104653

104654

104655

104656

104657

104658

104659

104660 STDERR

104661 The standard error shall be used only for diagnostic messages.

104662 OUTPUT FILES

104663 None.

104664 EXTENDED DESCRIPTION

104665 None.

104666 EXIT STATUS

104667 The following exit values shall be returned:

104668 0 Successful completion.

104669 >0 An error occurred.

104670 CONSEQUENCES OF ERRORS

104671 Default.

104672 APPLICATION USAGE

104673 Things can change while *ps* is running; the snapshot it gives is only true for an instant, and
104674 might not be accurate by the time it is displayed.

104675 The **args** format specifier is allowed to produce a truncated version of the command arguments.
104676 In some implementations, this information is no longer available when the *ps* utility is executed.

104677 If the field width is too narrow to display a textual ID, the system may use a numeric version.
104678 Normally, the system would be expected to choose large enough field widths, but if a large
104679 number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on
104680 one line. One way to ensure adequate width for the textual IDs is to override the default header
104681 for a field to make it larger than most or all user or group names.

104682 There is no special quoting mechanism for header text. The header text is the rest of the
104683 argument. If multiple header changes are needed, multiple **-o** options can be used, such as:

```
104684 ps -o "user=User Name" -o pid=Process\ ID
```

104685 On some implementations, especially multi-level secure systems, *ps* may be severely restricted
104686 and produce information only about child processes owned by the user.

104687 EXAMPLES

104688 The command:

```
104689 ps -o user,pid,ppid=MOM -o args
```

104690 writes at least the following in the POSIX locale:

```
104691 USER PID MOM COMMAND
```

104692 helene 34 12 ps -o uid,pid,ppid=MOM -o args

104693 The contents of the **COMMAND** field need not be the same in all implementations, due to
104694 possible truncation.

104695 RATIONALE

104696 There is very little commonality between BSD and System V implementations of *ps*. Many
104697 options conflict or have subtly different usages. The standard developers attempted to select a
104698 set of options for the base standard that were useful on a wide range of systems and selected
104699 options that either can be implemented on both BSD and System V-based systems without
104700 breaking the current implementations or where the options are sufficiently similar that any
104701 changes would not be unduly problematic for users or implementors.

104702 It is recognized that on some implementations, especially multi-level secure systems, *ps* may be
104703 nearly useless. The default output has therefore been chosen such that it does not break
104704 historical implementations and also is likely to provide at least some useful information on most
104705 systems.

104706 The major change is the addition of the format specification capability. The motivation for this
104707 invention is to provide a mechanism for users to access a wider range of system information, if
104708 the system permits it, in a portable manner. The fields chosen to appear in this volume of
104709 POSIX.1-2017 were arrived at after considering what concepts were likely to be both reasonably
104710 useful to the “average” user and had a reasonable chance of being implemented on a wide range
104711 of systems. Again it is recognized that not all systems are able to provide all the information
104712 and, conversely, some may wish to provide more. It is hoped that the approach adopted will be
104713 sufficiently flexible and extensible to accommodate most systems. Implementations may be
104714 expected to introduce new format specifiers.

104715 The default output should consist of a short listing containing the process ID, terminal name,
104716 cumulative execution time, and command name of each process.

104717 The preference of the standard developers would have been to make the format specification an
104718 operand of the *ps* command. Unfortunately, BSD usage precluded this.

104719 At one time a format was included to display the environment array of the process. This was
104720 deleted because there is no portable way to display it.

104721 The **-A** option is equivalent to the BSD **-g** and the SVID **-e**. Because the two systems differed, a
104722 mnemonic compromise was selected.

104723 The **-a** option is described with some optional behavior because the SVID omits session leaders,
104724 but BSD does not.

104725 In an early proposal, format specifiers appeared for priority and start time. The former was not
104726 defined adequately in this volume of POSIX.1-2017 and was removed in deference to the defined
104727 nice value; the latter because elapsed time was considered to be more useful.

104728 In a new BSD version of *ps*, a **-O** option can be used to write all of the default information,
104729 followed by additional format specifiers. This was not adopted because the default output is
104730 implementation-defined. Nevertheless, this is a useful option that should be reserved for that
104731 purpose. In the **-o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of
104732 each **-o**. Therefore, the user can have an alias or function that defines the beginning of their
104733 desired format and add more fields to the end of the output in certain cases where that would be
104734 useful.

104735 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
104736 require that they all use the same format.

104737 The **pcpu** field indicates that the CPU time available is determined in an unspecified manner.
104738 This is because it is difficult to express an algorithm that is useful across all possible machine
104739 architectures. Historical counterparts to this value have attempted to show percentage of use in
104740 the recent past, such as the preceding minute. Frequently, these values for all processes did not
104741 add up to 100%. Implementations are encouraged to provide data in this field to users that will
104742 help them identify processes currently affecting the performance of the system.

104743 **FUTURE DIRECTIONS**

104744 None.

104745 **SEE ALSO**

104746 *kill, nice, renice*

104747 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

104748 **CHANGE HISTORY**

104749 First released in Issue 2.

104750 **Issue 6**

104751 This utility is marked as part of the User Portability Utilities option.

104752 The normative text is reworded to avoid use of the term “must” for application requirements.

104753 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

104754 **Issue 7**

104755 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104756 SD5-XCU-ERN-148 is applied, updating the OPTIONS section.

104757 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0160 [584] is applied.

104758 **NAME**

104759 pwd — return working directory name

104760 **SYNOPSIS**

104761 pwd [-L|-P]

104762 **DESCRIPTION**104763 The *pwd* utility shall write to standard output an absolute pathname of the current working
104764 directory, which does not contain the filenames dot or dot-dot.104765 **OPTIONS**104766 The *pwd* utility shall conform to XBD [Section 12.2](#) (on page 216).

104767 The following options shall be supported by the implementation:

104768 **-L** If the *PWD* environment variable contains an absolute pathname of the current
104769 directory and the pathname does not contain any components that are dot or dot-
104770 dot, *pwd* shall write this pathname to standard output, except that if the *PWD*
104771 environment variable is longer than {PATH_MAX} bytes including the terminating
104772 null, it is unspecified whether *pwd* writes this pathname to standard output or
104773 behaves as if the **-P** option had been specified. Otherwise, the **-L** option shall
104774 behave as the **-P** option.

104775 **-P** The pathname written to standard output shall not contain any components that
104776 refer to files of type symbolic link. If there are multiple pathnames that the *pwd*
104777 utility could write to standard output, one beginning with a single <slash>
104778 character and one or more beginning with two <slash> characters, then it shall
104779 write the pathname beginning with a single <slash> character. The pathname shall
104780 not contain any unnecessary <slash> characters after the leading one or two
104781 <slash> characters.

104782 If both **-L** and **-P** are specified, the last one shall apply. If neither **-L** nor **-P** is specified, the *pwd*
104783 utility shall behave as if **-L** had been specified.

104784 **OPERANDS**

104785 None.

104786 **STDIN**

104787 Not used.

104788 **INPUT FILES**

104789 None.

104790 **ENVIRONMENT VARIABLES**104791 The following environment variables shall affect the execution of *pwd*:

104792 **LANG** Provide a default value for the internationalization variables that are unset or null.
104793 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
104794 used to determine the values of locale categories.)

104795 **LC_ALL** If set to a non-empty string value, override the values of all the other
104796 internationalization variables.

104797 **LC_MESSAGES**

104798 Determine the locale that should be used to affect the format and contents of
104799 diagnostic messages written to standard error.

104800 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

104801 *PWD* An absolute pathname of the current working directory. If an application sets or
104802 unsets the value of *PWD*, the behavior of *pwd* is unspecified.

104803 **ASYNCHRONOUS EVENTS**

104804 Default.

104805 **STDOUT**

104806 The *pwd* utility output is an absolute pathname of the current working directory:

104807 "%s\n", <directory pathname>

104808 **STDERR**

104809 The standard error shall be used only for diagnostic messages.

104810 **OUTPUT FILES**

104811 None.

104812 **EXTENDED DESCRIPTION**

104813 None.

104814 **EXIT STATUS**

104815 The following exit values shall be returned:

104816 0 Successful completion.

104817 >0 An error occurred.

104818 **CONSEQUENCES OF ERRORS**

104819 If an error is detected, output shall not be written to standard output, a diagnostic message shall
104820 be written to standard error, and the exit status is not zero.

104821 **APPLICATION USAGE**

104822 If the pathname obtained from *pwd* is longer than {PATH_MAX} bytes, it could produce an error
104823 if passed to *cd*. Therefore, in order to return to that directory it may be necessary to break the
104824 pathname into sections shorter than {PATH_MAX} and call *cd* on each section in turn (the first
104825 section being an absolute pathname and subsequent sections being relative pathnames).

104826 **EXAMPLES**

104827 None.

104828 **RATIONALE**

104829 Some implementations have historically provided *pwd* as a shell special built-in command.

104830 In most utilities, if an error occurs, partial output may be written to standard output. This does
104831 not happen in historical implementations of *pwd*. Because *pwd* is frequently used in historical
104832 shell scripts without checking the exit status, it is important that the historical behavior is
104833 required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any
104834 partial output being written to standard output.

104835 An earlier version of this standard stated that the *PWD* environment variable was affected when
104836 the *-P* option was in effect. This was incorrect; conforming implementations do not do this.

104837 **FUTURE DIRECTIONS**

104838 None.

104839 **SEE ALSO**

104840 *cd*

104841 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

104842 XSH [getcwd\(\)](#)

104843 **CHANGE HISTORY**

104844 First released in Issue 2.

104845 **Issue 6**

104846 The **-P** and **-L** options are added to describe actions relating to symbolic links as specified in the
104847 IEEE P1003.2b draft standard.

104848 **Issue 7**

104849 Austin Group Interpretation 1003.1-2001 #097 is applied.

104850 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104851 Changes to the *pwd* utility and *PWD* environment variable have been made to match the
104852 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.

104853 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0161 [471] is applied.

104854 **NAME**104855 qalter `q'alter batch job`104856 **SYNOPSIS**

```

104857 OB BE qalter [-a date_time] [-A account_string] [-c interval] [-e path_name]
104858          [-h hold_list] [-j join_list] [-k keep_list] [-l resource_list]
104859          [-m mail_options] [-M mail_list] [-N name] [-o path_name]
104860          [-p priority] [-r y|n] [-S path_name_list] [-u user_list]
104861          job_identifier...
```

104862 **DESCRIPTION**

104863 The attributes of a batch job are altered by a request to the batch server that manages the batch
 104864 job. The *qalter* utility is a user-accessible batch client that requests the alteration of the attributes
 104865 of one or more batch jobs.

104866 The *qalter* utility shall alter the attributes of those batch jobs, and only those batch jobs, for which
 104867 a batch *job_identifier* is presented to the utility.

104868 The *qalter* utility shall alter the attributes of batch jobs in the order in which the batch
 104869 *job_identifiers* are presented to the utility.

104870 If the *qalter* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 104871 process the remaining batch *job_identifiers*, if any.

104872 For each batch *job_identifier* for which the *qalter* utility succeeds, each attribute of the identified
 104873 batch job shall be altered as indicated by all the options presented to the utility.

104874 For each identified batch job for which the *qalter* utility fails, the utility shall not alter any
 104875 attribute of the batch job.

104876 For each batch job that the *qalter* utility processes, the utility shall not modify any attribute other
 104877 than those required by the options and option-arguments presented to the utility.

104878 The *qalter* utility shall alter batch jobs by sending a *Modify Job Request* to the batch server that
 104879 manages each batch job. At the time the *qalter* utility exits, it shall have modified the batch job
 104880 corresponding to each successfully processed batch *job_identifier*. An attempt to alter the
 104881 attributes of a batch job in the RUNNING state is implementation-defined.

104882 **OPTIONS**

104883 The *qalter* utility shall conform to XBD [Section 12.2](#) (on page 216).

104884 The following options shall be supported by the implementation:

104885 **-a date_time** Redefine the time at which the batch job becomes eligible for execution.

104886 The *date_time* argument shall be in the same form and represent the same time as
 104887 for the *touch* utility. The time so represented shall be set into the *Execution_Time*
 104888 attribute of the batch job. If the time specified is earlier than the current time, the
 104889 **-a** option shall have no effect.

104890 **-A account_string**

104891 Redefine the account to which the resource consumption of the batch job should be
 104892 charged.

104893 The syntax of the *account_string* option-argument is unspecified.

104894 The *qalter* utility shall set the *Account_Name* attribute of the batch job to the value
 104895 of the *account_string* option-argument.

104896 **-c interval** Redefine whether the batch job should be checkpointed, and if so, how often.

104897 The *qalter* utility shall accept a value for the interval option-argument that is one of

104898 the following:

104899 n No checkpointing is to be performed on the batch job

104900 (NO_CHECKPOINT).

104901 s Checkpointing is to be performed only when the batch server is shut

104902 down (CHECKPOINT_AT_SHUTDOWN).

104903 c Automatic periodic checkpointing is to be performed at the

104904 *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU

104905 minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).

104906 c=*minutes* Automatic periodic checkpointing is to be performed every *minutes*

104907 of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is

104908 greater. The *minutes* argument shall conform to the syntax for

104909 unsigned integers and shall be greater than zero.

104910 An implementation may define other checkpoint intervals. The conformance

104911 document for an implementation shall describe any alternative checkpoint

104912 intervals, how they are specified, their internal behavior, and how they affect the

104913 behavior of the utility.

104914 The *qalter* utility shall set the *Checkpoint* attribute of the batch job to the value of the

104915 *interval* option-argument.

104916 **-e path_name** Redefine the path to be used for the standard error stream of the batch job.

104917 The *qalter* utility shall accept a *path_name* option-argument that conforms to the

104918 syntax of the *path_name* element defined in the System Interfaces volume of

104919 POSIX.1-2017, which can be preceded by a host name element of the form

104920 *hostname*:.
104921 *hostname*..

104922 If the *path_name* option-argument constitutes an absolute pathname, the *qalter*

104923 utility shall set the *Error_Path* attribute of the batch job to the value of the

104924 *path_name* option-argument, including the host name element, if present.

104925 If the *path_name* option-argument constitutes a relative pathname and no host

104926 name element is specified, the *qalter* utility shall set the *Error_Path* attribute of the

104927 batch job to the value of the absolute pathname derived by expanding the

104928 *path_name* option-argument relative to the current directory of the process that

104929 executes the *qalter* utility.

104930 If the *path_name* option-argument constitutes a relative pathname and a host name

104931 element is specified, the *qalter* utility shall set the *Error_Path* attribute of the batch

104932 job to the value of the option-argument without expansion.

104933 If the *path_name* option-argument does not include a host name element, the *qalter*

104934 utility shall prefix the pathname in the *Error_Path* attribute with *hostname*:, where

104935 *hostname* is the name of the host upon which the *qalter* utility is being executed.

104936 **-h hold_list** Redefine the types of holds, if any, on the batch job. The *qalter* **-h** option shall

104937 accept a value for the *hold_list* option-argument that is a string of alphanumeric

104938 characters in the portable character set.

104939 The *qalter* utility shall accept a value for the *hold_list* option-argument that is a

104940 string of one or more of the characters 'u', 's', or 'o', or the single character
 104941 'n'. For each unique character in the *hold_list* option-argument, the *qalter* utility
 104942 shall add a value to the *Hold_Types* attribute of the batch job as follows, each
 104943 representing a different hold type:

- 104944 u USER
- 104945 s SYSTEM
- 104946 o OPERATOR

104947 If any of these characters are duplicated in the *hold_list* option-argument, the
 104948 duplicates shall be ignored. An existing *Hold_Types* attribute can be cleared by the
 104949 hold type:

- 104950 n NO_HOLD

104951 The *qalter* utility shall consider it an error if any hold type other than 'n' is
 104952 combined with hold type 'n'. Strictly conforming applications shall not repeat
 104953 any of the characters 'u', 's', 'o', or 'n' within the *hold_list* option-argument.
 104954 The *qalter* utility shall permit the repetition of characters, but shall not assign
 104955 additional meaning to the repeated characters. An implementation may define
 104956 other hold types. The conformance document for an implementation shall describe
 104957 any additional hold types, how they are specified, their internal behavior, and how
 104958 they affect the behavior of the utility.

104959 **-j *join_list*** Redefine which streams of the batch job are to be merged. The *qalter* **-j** option shall
 104960 accept a value for the *join_list* option-argument that is a string of alphanumeric
 104961 characters in the portable character set.

104962 The *qalter* utility shall accept a *join_list* option-argument that consists of one or
 104963 more of the characters 'e' and 'o', or the single character 'n'.

104964 All of the other batch job output streams specified shall be merged into the output
 104965 stream represented by the character listed first in the *join_list* option-argument.

104966 For each unique character in the *join_list* option-argument, the *qalter* utility shall
 104967 add a value to the *Join_Path* attribute of the batch job as follows, each representing
 104968 a different batch job stream to join:

- 104969 e The standard error of the batch job (JOIN_STD_ERROR).
- 104970 o The standard output of the batch job (JOIN_STD_OUTPUT).

104971 An existing *Join_Path* attribute can be cleared by the join type:

- 104972 n NO_JOIN

104973 If 'n' is specified, then no files are joined. The *qalter* utility shall consider it an
 104974 error if any join type other than 'n' is combined with join type 'n'.

104975 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 104976 'n' within the *join_list* option-argument. The *qalter* utility shall permit the
 104977 repetition of characters, but shall not assign additional meaning to the repeated
 104978 characters.

104979 An implementation may define other join types. The conformance document for an
 104980 implementation shall describe any additional batch job streams, how they are
 104981 specified, their internal behavior, and how they affect the behavior of the utility.

104982 **-k** *keep_list* Redefine which output of the batch job to retain on the execution host.

104983 The *qalter -k* option shall accept a value for the *keep_list* option-argument that is a
104984 string of alphanumeric characters in the portable character set.

104985 The *qalter* utility shall accept a *keep_list* option-argument that consists of one or
104986 more of the characters 'e' and 'o', or the single character 'n'.

104987 For each unique character in the *keep_list* option-argument, the *qalter* utility shall
104988 add a value to the *Keep_Files* attribute of the batch job as follows, each representing
104989 a different batch job stream to keep:

104990 e The standard error of the batch job (KEEP_STD_ERROR).

104991 o The standard output of the batch job (KEEP_STD_OUTPUT).

104992 If both 'e' and 'o' are specified, then both files are retained. An existing
104993 *Keep_Files* attribute can be cleared by the keep type:

104994 n NO_KEEP

104995 If 'n' is specified, then no files are retained. The *qalter* utility shall consider it an
104996 error if any keep type other than 'n' is combined with keep type 'n'.

104997 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
104998 'n' within the *keep_list* option-argument. The *qalter* utility shall permit the
104999 repetition of characters, but shall not assign additional meaning to the repeated
105000 characters. An implementation may define other keep types. The conformance
105001 document for an implementation shall describe any additional keep types, how
105002 they are specified, their internal behavior, and how they affect the behavior of the
105003 utility.

105004 **-l** *resource_list* Redefine the resources that are allowed or required by the batch job.

105005 The *qalter* utility shall accept a *resource_list* option-argument that conforms to the
105006 following syntax:

105007 resource=value[, , resource=value , , ...]

105008 The *qalter* utility shall set one entry in the value of the *Resource_List* attribute of the
105009 batch job for each resource listed in the *resource_list* option-argument.

105010 Because the list of supported resource names might vary by batch server, the *qalter*
105011 utility shall rely on the batch server to validate the resource names and associated
105012 values. See [Section 3.3.3](#) (on page 2451) for a means of removing *keyword=value*
105013 (and *value@keyword*) pairs and other general rules for list-oriented batch job
105014 attributes.
105015

105016 **-m** *mail_options* Redefine the points in the execution of the batch job at which the batch server is to
105017 send mail about a change in the state of the batch job.

105018 The *qalter -m* option shall accept a value for the *mail_options* option-argument that
105019 is a string of alphanumeric characters in the portable character set.

105020 The *qalter* utility shall accept a value for the *mail_options* option-argument that is a
105021 string of one or more of the characters 'e', 'b', and 'a', or the single character
105022 'n'. For each unique character in the *mail_options* option-argument, the *qalter*
105023 utility shall add a value to the *Mail_Users* attribute of the batch job as follows, each
105024

105025 representing a different time during the life of a batch job at which to send mail:

105026 e MAIL_AT_EXIT

105027 b MAIL_AT_BEGINNING

105028 a MAIL_AT_ABORT

105029 If any of these characters are duplicated in the *mail_options* option-argument, the

105030 duplicates shall be ignored.

105031 An existing *Mail_Points* attribute can be cleared by the mail type:

105032 n NO_MAIL

105033 If 'n' is specified, then mail is not sent. The *qalter* utility shall consider it an error

105034 if any mail type other than 'n' is combined with mail type 'n'. Strictly

105035 conforming applications shall not repeat any of the characters 'e', 'b', 'a', or

105036 'n' within the *mail_options* option-argument. The *qalter* utility shall permit the

105037 repetition of characters but shall not assign additional meaning to the repeated

105038 characters.

105039 An implementation may define other mail types. The conformance document for

105040 an implementation shall describe any additional mail types, how they are

105041 specified, their internal behavior, and how they affect the behavior of the utility.

105042 **-M** *mail_list* Redefine the list of users to which the batch server that executes the batch job is to

105043 send mail, if the batch server sends mail about the batch job.

105044 The syntax of the *mail_list* option-argument is unspecified. If the implementation

105045 of the *qalter* utility uses a name service to locate users, the utility shall accept the

105046 syntax used by the name service.

105047 If the implementation of the *qalter* utility does not use a name service to locate

105048 users, the implementation shall accept the following syntax for user names:

105049 `mail_address[, ,mail_address, , . . .]`

105050 The interpretation of *mail_address* is implementation-defined.

105051 The *qalter* utility shall set the *Mail_Users* attribute of the batch job to the value of

105052 the *mail_list* option-argument.

105053 **-N** *name* Redefine the name of the batch job.

105054 The *qalter* **-N** option shall accept a value for the *name* option-argument that is a

105055 string of up to 15 alphanumeric characters in the portable character set where the

105056 first character is alphabetic.

105057 The syntax of the *name* option-argument is unspecified.

105058 The *qalter* utility shall set the *Job_Name* attribute of the batch job to the value of the

105059 *name* option-argument.

105060 **-o** *path_name*

105061 Redefine the path for the standard output of the batch job.

105062 The *qalter* utility shall accept a *path_name* option-argument that conforms to the

105063 syntax of the *path_name* element defined in the System Interfaces volume of

105064 POSIX.1-2017, which can be preceded by a host name element of the form

105065 *hostname*..

105066 If the *path_name* option-argument constitutes an absolute pathname, the *qalter*
 105067 utility shall set the *Output_Path* attribute of the batch job to the value of the
 105068 *path_name* option-argument.

105069 If the *path_name* option-argument constitutes a relative pathname and no host
 105070 name element is specified, the *qalter* utility shall set the *Output_Path* attribute of the
 105071 batch job to the absolute pathname derived by expanding the *path_name* option-
 105072 argument relative to the current directory of the process that executes the *qalter*
 105073 utility.

105074 If the *path_name* option-argument constitutes a relative pathname and a host name
 105075 element is specified, the *qalter* utility shall set the *Output_Path* attribute of the batch
 105076 job to the value of the *path_name* option-argument without any expansion of the
 105077 pathname.

105078 If the *path_name* option-argument does not include a host name element, the *qalter*
 105079 utility shall prefix the pathname in the *Output_Path* attribute with *hostname:*, where
 105080 *hostname* is the name of the host upon which the *qalter* utility is being executed.

105081 **-p** *priority* Redefine the priority of the batch job.

105082 The *qalter* utility shall accept a value for the priority option-argument that
 105083 conforms to the syntax for signed decimal integers, and which is not less than
 105084 -1 024 and not greater than 1 023.

105085 The *qalter* utility shall set the *Priority* attribute of the batch job to the value of the
 105086 *priority* option-argument.

105087 **-r** *y | n* Redefine whether the batch job is rerunnable.

105088 If the value of the option-argument is 'y', the *qalter* utility shall set the *Rerunable*
 105089 attribute of the batch job to TRUE.

105090 If the value of the option-argument is 'n', the *qalter* utility shall set the *Rerunable*
 105091 attribute of the batch job to FALSE.

105092 The *qalter* utility shall consider it an error if any character other than 'y' or 'n' is
 105093 specified in the option-argument.

105094 **-S** *path_name_list* Redefine the shell that interprets the script at the destination system.

105095 The *qalter* utility shall accept a *path_name_list* option-argument that conforms to the
 105096 following syntax:
 105097

```
pathname[ @host ] [ , pathname[ @host ] , . . . ]
```

105098 The *qalter* utility shall accept only one pathname that is missing a corresponding
 105099 host name. The *qalter* utility shall allow only one pathname per named host.

105100 The *qalter* utility shall add a value to the *Shell_Path_List* attribute of the batch job
 105101 for each entry in the *path_name_list* option-argument. See [Section 3.3.3](#) (on page
 105102 2451) for a means of removing *keyword=value* (and *value@keyword*) pairs and other
 105103 general rules for list-oriented batch job attributes.

105104

105105 **-u** *user_list* Redefine the user name under which the batch job is to run at the destination
 105106 system.

105107 The *qalter* utility shall accept a *user_list* option-argument that conforms to the
 105108 following syntax:

- 105109 username[@host][, ,username[@host], , . . .]
- 105110 The *qalter* utility shall accept only one user name that is missing a corresponding
105111 host name. The *qalter* utility shall accept only one user name per named host.
- 105112 The *qalter* utility shall add a value to the *User_List* attribute of the batch job for each
105113 entry in the *user_list* option-argument. See [Section 3.3.3](#) (on page 2451) for a means
105114 of removing *keyword=value* (and *value@keyword*) pairs and other general rules for
105115 list-oriented batch job attributes.
- 105116 **OPERANDS**
- 105117 The *qalter* utility shall accept one or more operands that conform to the syntax for a batch
105118 *job_identifier* (see [Section 3.3.1](#), on page 2449).
- 105119 **STDIN**
- 105120 Not used.
- 105121 **INPUT FILES**
- 105122 None.
- 105123 **ENVIRONMENT VARIABLES**
- 105124 The following environment variables shall affect the execution of *qalter*:
- 105125 *LANG* Provide a default value for the internationalization variables that are unset or null.
105126 (See [XBD Section 8.2](#) (on page 174) the precedence of internationalization variables
105127 used to determine the values of locale categories.)
- 105128 *LC_ALL* If set to a non-empty string value, override the values of all the other
105129 internationalization variables.
- 105130 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
105131 characters (for example, single-byte as opposed to multi-byte characters in
105132 arguments).
- 105133 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
105134 diagnostic messages written to standard error.
105135
- 105136 *LOGNAME* Determine the login name of the user.
- 105137 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
105138 unset or null, an unspecified default timezone shall be used.
- 105139 **ASYNCHRONOUS EVENTS**
- 105140 Default.
- 105141 **STDOUT**
- 105142 None.
- 105143 **STDERR**
- 105144 The standard error shall be used only for diagnostic messages.
- 105145 **OUTPUT FILES**
- 105146 None.
- 105147 **EXTENDED DESCRIPTION**
- 105148 None.

105149 **EXIT STATUS**

105150 The following exit values shall be returned:

105151 0 Successful completion.

105152 >0 An error occurred.

105153 **CONSEQUENCES OF ERRORS**

105154 In addition to the default behavior, the *qalter* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qalter* utility attempts to locate the batch job on other batch servers is implementation-defined.

105158 **APPLICATION USAGE**

105159 None.

105160 **EXAMPLES**

105161 None.

105162 **RATIONALE**

105163 The *qalter* utility allows users to change the attributes of a batch job.

105164 As a means of altering a queued job, the *qalter* utility is superior to deleting and requeuing the batch job insofar as an altered job retains its place in the queue with some traditional selection algorithms. In addition, the *qalter* utility is both shorter and simpler than a sequence of *qdel* and *qsub* utilities.

105168 The result of an attempt on the part of a user to alter a batch job in a RUNNING state is implementation-defined because a batch job in the RUNNING state will already have opened its output files and otherwise performed any actions indicated by the options in effect at the time the batch job began execution.

105172 The options processed by the *qalter* utility are identical to those of the *qsub* utility, with a few exceptions: **-V**, **-v**, and **-q**. The **-V** and **-v** are inappropriate for the *qalter* utility, since they capture potentially transient environment information from the submitting process. The **-q** option would specify a new queue, which would largely negate the previously stated advantage of using *qalter*; furthermore, the *qmove* utility provides a superior means of moving jobs.

105177 Each of the following paragraphs provides the rationale for a *qalter* option.

105178 Additional rationale concerning these options can be found in the rationale for the *qsub* utility.

105179 The **-a** option allows users to alter the date and time at which a batch job becomes eligible to run.

105181 The **-A** option allows users to change the account that will be charged for the resources consumed by the batch job. Support for the **-A** option is mandatory for conforming implementations of *qalter*, even though support of accounting is optional for servers. Whether or not to support accounting is left to the implementor of the server, but mandatory support of the **-A** option assures users of a consistent interface and allows them to control accounting on servers that support accounting.

105187 The **-c** option allows users to alter the checkpointing interval of a batch job. A checkpointing system, which is not defined by POSIX.1-2017, allows recovery of a batch job at the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume expensive computing time or must meet a critical schedule. Users should be allowed to make the tradeoff between the overhead of checkpointing and the risk to the timely completion of the batch job; therefore, this volume of POSIX.1-2017 provides the checkpointing interval option. Support for checkpointing is optional for servers.

- 105194 The **-e** option allows users to alter the name and location of the standard error stream written by
105195 a batch job. However, the path of the standard error stream is meaningless if the value of the
105196 *Join_Path* attribute of the batch job is TRUE.
- 105197 The **-h** option allows users to set the hold type in the *Hold_Types* attribute of a batch job. The
105198 *qhold* and *qrls* utilities add or remove hold types to the *Hold_Types* attribute, respectively. The **-h**
105199 option has been modified to allow for implementation-defined hold types.
- 105200 The **-j** option allows users to alter the decision to join (merge) the standard error stream of the
105201 batch job with the standard output stream of the batch job.
- 105202 The **-l** option allows users to change the resource limits imposed on a batch job.
- 105203 The **-m** option allows users to modify the list of points in the life of a batch job at which the
105204 designated users will receive mail notification.
- 105205 The **-M** option allows users to alter the list of users who will receive notification about events in
105206 the life of a batch job.
- 105207 The **-N** option allows users to change the name of a batch job.
- 105208 The **-o** option allows users to alter the name and path to which the standard output stream of
105209 the batch job will be written.
- 105210 The **-P** option allows users to modify the priority of a batch job. Support for priority is optional
105211 for batch servers.
- 105212 The **-r** option allows users to alter the rerunability status of a batch job.
- 105213 The **-S** option allows users to change the name and location of the shell image that will be
105214 invoked to interpret the script of the batch job. This option has been modified to allow a list of
105215 shell name and locations associated with different hosts.
- 105216 The **-u** option allows users to change the user identifier under which the batch job will execute.
- 105217 The *job_identifier* operand syntax is provided so that the user can differentiate between the
105218 originating and destination (or executing) batch server. These may or may not be the same. The
105219 *.server_name* portion identifies the originating batch server, while the *@server* portion identifies
105220 the destination batch server.
- 105221 Historically, the *qalter* utility has been a component of the Network Queuing System (NQS), the
105222 existing practice from which this utility has been derived.
- 105223 **FUTURE DIRECTIONS**
- 105224 The *qalter* utility may be removed in a future version.
- 105225 **SEE ALSO**
- 105226 [Chapter 3](#) (on page 2427), *qdel*, *qhold*, *qmove*, *qrls*, *qsub*, *touch*
- 105227 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 105228 **CHANGE HISTORY**
- 105229 Derived from IEEE Std 1003.2d-1994.
- 105230 **Issue 6**
- 105231 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.
- 105232 IEEE PASC Interpretation 1003.2 #182 is applied, clarifying the description of the **-a** option.

105233 **Issue 7**

105234 The *qalter* utility is marked obsolescent.

105235 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105236 **NAME**

105237 qdel ‡delete batch jobs

105238 **SYNOPSIS**105239 OB BE qdel *job_identifier...*105240 **DESCRIPTION**

105241 A batch job is deleted by sending a request to the batch server that manages the batch job. A
 105242 batch job that has been deleted is no longer subject to management by batch services.

105243 The *qdel* utility is a user-accessible client of batch services that requests the deletion of one or
 105244 more batch jobs.

105245 The *qdel* utility shall request a batch server to delete those batch jobs for which a batch
 105246 *job_identifier* is presented to the utility.

105247 The *qdel* utility shall delete batch jobs in the order in which their batch *job_identifiers* are
 105248 presented to the utility.

105249 If the *qdel* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
 105250 process the remaining batch *job_identifiers*, if any.

105251 The *qdel* utility shall delete each batch job by sending a *Delete Job Request* to the batch server that
 105252 manages the batch job.

105253 The *qdel* utility shall not exit until the batch job corresponding to each successfully processed
 105254 batch *job_identifier* has been deleted.

105255 **OPTIONS**

105256 None.

105257 **OPERANDS**

105258 The *qdel* utility shall accept one or more operands that conform to the syntax for a batch
 105259 *job_identifier* (see [Section 3.3.1](#), on page 2449).

105260 **STDIN**

105261 Not used.

105262 **INPUT FILES**

105263 None.

105264 **ENVIRONMENT VARIABLES**105265 The following environment variables shall affect the execution of *qdel*:

105266 *LANG* Provide a default value for the internationalization variables that are unset or null.
 105267 (See [XBD Section 8.2](#) (on page 174) the precedence of internationalization variables
 105268 used to determine the values of locale categories.)

105269 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105270 internationalization variables.

105271 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105272 characters (for example, single-byte as opposed to multi-byte characters in
 105273 arguments).

105274 *LC_MESSAGES*

105275 Determine the locale that should be used to affect the format and contents of
 105276 diagnostic messages written to standard error.

- 105277 *LOGNAME* Determine the login name of the user.
- 105278 **ASYNCHRONOUS EVENTS**
- 105279 Default.
- 105280 **STDOUT**
- 105281 An implementation of the *qdel* utility may write informative messages to standard output.
- 105282 **STDERR**
- 105283 The standard error shall be used only for diagnostic messages.
- 105284 **OUTPUT FILES**
- 105285 None.
- 105286 **EXTENDED DESCRIPTION**
- 105287 None.
- 105288 **EXIT STATUS**
- 105289 The following exit values shall be returned:
- 105290 0 Successful completion.
- 105291 >0 An error occurred.
- 105292 **CONSEQUENCES OF ERRORS**
- 105293 In addition to the default behavior, the *qdel* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qdel* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.
- 105294
- 105295
- 105296
- 105297
- 105298 **APPLICATION USAGE**
- 105299 None.
- 105300 **EXAMPLES**
- 105301 None.
- 105302 **RATIONALE**
- 105303 The *qdel* utility allows users and administrators to delete jobs.
- 105304 The *qdel* utility provides functionality that is not otherwise available. For example, the *kill* utility of the operating system does not suffice. First, to use the *kill* utility, the user might have to log in on a remote node, because the *kill* utility does not operate across the network. Second, unlike *qdel*, *kill* cannot remove jobs from queues. Lastly, the arguments of the *qdel* utility are job identifiers rather than process identifiers, and so this utility can be passed the output of the *qselect* utility, thus providing users with a means of deleting a list of jobs.
- 105305
- 105306
- 105307
- 105308
- 105309
- 105310 Because a set of jobs can be selected using the *qselect* utility, the *qdel* utility has not been complicated with options that provide for selection of jobs. Instead, the batch jobs to be deleted are identified individually by their job identifiers.
- 105311
- 105312
- 105313 Historically, the *qdel* utility has been a component of NQS, the existing practice on which it is based. However, the *qdel* utility defined in this volume of POSIX.1-2017 does not provide an option for specifying a signal number to send to the batch job prior to the killing of the process; that capability has been subsumed by the *qsig* utility.
- 105314
- 105315
- 105316
- 105317 A discussion was held about the delays of networking and the possibility that the batch server may never respond, due to a down router, down batch server, or other network mishap. The DESCRIPTION records this under the words “fails to process any job identifier”. In the broad sense, the network problem is also an error, which causes the failure to process the batch job
- 105318
- 105319
- 105320

- 105321 identifier.
- 105322 **FUTURE DIRECTIONS**
- 105323 The *qdel* utility may be removed in a future version.
- 105324 **SEE ALSO**
- 105325 [Chapter 3](#) (on page 2427), *kill*, *qselect*, *qsig*
- 105326 XBD [Chapter 8](#) (on page 173)
- 105327 **CHANGE HISTORY**
- 105328 Derived from IEEE Std 1003.2d-1994.
- 105329 **Issue 6**
- 105330 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.
- 105331 **Issue 7**
- 105332 The *qdel* utility is marked obsolescent.
- 105333 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105334 **NAME**

105335 qhold ‡hold batch jobs

105336 **SYNOPSIS**105337 OB BE qhold [-h *hold_list*] *job_identifier...*105338 **DESCRIPTION**

105339 A hold is placed on a batch job by a request to the batch server that manages the batch job. A
 105340 batch job that has one or more holds is not eligible for execution. The *qhold* utility is a user-
 105341 accessible client of batch services that requests one or more types of hold to be placed on one or
 105342 more batch jobs.

105343 The *qhold* utility shall place holds on those batch jobs for which a batch *job_identifier* is presented
 105344 to the utility.

105345 The *qhold* utility shall place holds on batch jobs in the order in which their batch *job_identifiers*
 105346 are presented to the utility. If the *qhold* utility fails to process any batch *job_identifier* successfully,
 105347 the utility shall proceed to process the remaining batch *job_identifiers*, if any.

105348 The *qhold* utility shall place holds on each batch job by sending a *Hold Job Request* to the batch
 105349 server that manages the batch job.

105350 The *qhold* utility shall not exit until holds have been placed on the batch job corresponding to
 105351 each successfully processed batch *job_identifier*.

105352 **OPTIONS**

105353 The *qhold* utility shall conform to XBD [Section 12.2](#) (on page 216).

105354 The following option shall be supported by the implementation:

105355 **-h *hold_list*** Define the types of holds to be placed on the batch job.

105356 The *qhold* **-h** option shall accept a value for the *hold_list* option-argument that is a
 105357 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 105358 [6.1](#), on page 125).

105359 The *qhold* utility shall accept a value for the *hold_list* option-argument that is a
 105360 string of one or more of the characters 'u', 's', or 'o', or the single character
 105361 'n'.

105362 For each unique character in the *hold_list* option-argument, the *qhold* utility shall
 105363 add a value to the *Hold_Types* attribute of the batch job as follows, each
 105364 representing a different hold type:

105365 u USER

105366 s SYSTEM

105367 o OPERATOR

105368 If any of these characters are duplicated in the *hold_list* option-argument, the
 105369 duplicates shall be ignored.

105370 An existing *Hold_Types* attribute can be cleared by the following hold type:

105371 n NO_HOLD

105372 The *qhold* utility shall consider it an error if any hold type other than 'n' is
 105373 combined with hold type 'n'.

105374 Strictly conforming applications shall not repeat any of the characters 'u', 's',

105375 'o', or 'n' within the *hold_list* option-argument. The *qhold* utility shall permit the
 105376 repetition of characters, but shall not assign additional meaning to the repeated
 105377 characters.

105378 An implementation may define other hold types. The conformance document for
 105379 an implementation shall describe any additional hold types, how they are
 105380 specified, their internal behavior, and how they affect the behavior of the utility.

105381 If the **-h** option is not presented to the *qhold* utility, the implementation shall set
 105382 the *Hold_Types* attribute to USER.

105383 OPERANDS

105384 The *qhold* utility shall accept one or more operands that conform to the syntax for a batch
 105385 *job_identifier* (see [Section 3.3.1](#), on page 2449).

105386 STDIN

105387 Not used.

105388 INPUT FILES

105389 None.

105390 ENVIRONMENT VARIABLES

105391 The following environment variables shall affect the execution of *qhold*:

105392 *LANG* Provide a default value for the internationalization variables that are unset or null.
 105393 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
 105394 used to determine the values of locale categories.)

105395 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105396 internationalization variables.

105397 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105398 characters (for example, single-byte as opposed to multi-byte characters in
 105399 arguments).

105400 *LC_MESSAGES*

105401 Determine the locale that should be used to affect the format and contents of
 105402 diagnostic messages written to standard error.

105403 *LOGNAME* Determine the login name of the user.

105404 ASYNCHRONOUS EVENTS

105405 Default.

105406 STDOUT

105407 None.

105408 STDERR

105409 The standard error shall be used only for diagnostic messages.

105410 OUTPUT FILES

105411 None.

105412 EXTENDED DESCRIPTION

105413 None.

105414 EXIT STATUS

105415 The following exit values shall be returned:

105416 0 Successful completion.

105417 >0 An error occurred.

105418 CONSEQUENCES OF ERRORS

105419 In addition to the default behavior, the *qhold* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qhold* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

105424 APPLICATION USAGE

105425 None.

105426 EXAMPLES

105427 None.

105428 RATIONALE

105429 The *qhold* utility allows users to place a hold on one or more jobs. A hold makes a batch job ineligible for execution.

105431 The *qhold* utility has options that allow the user to specify the type of hold. Should the user wish to place a hold on a set of jobs that meet a selection criteria, such a list of jobs can be acquired using the *qselect* utility.

105434 The **-h** option allows the user to specify the type of hold that is to be placed on the job. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The USER and OPERATOR holds are distinct. The batch server that manages the batch job will verify that the user is authorized to set the specified hold for the batch job.

105438 Mail is not required on hold because the administrator has the tools and libraries to build this option if he or she wishes.

105440 Historically, the *qhold* utility has been a part of some existing batch systems, although it has not traditionally been a part of the NQS.

105442 FUTURE DIRECTIONS

105443 The *qhold* utility may be removed in a future version.

105444 SEE ALSO

105445 [Chapter 3](#) (on page 2427), *qselect*

105446 [XBD Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

105447 CHANGE HISTORY

105448 Derived from IEEE Std 1003.2d-1994.

105449 Issue 6

105450 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

105451 Issue 7

105452 The *qhold* utility is marked obsolescent.

105453 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105454 **NAME**

105455 qmove ‡'move batch jobs

105456 **SYNOPSIS**105457 OB BE `qmove destination job_identifier...`105458 **DESCRIPTION**

105459 To move a batch job is to remove the batch job from the batch queue in which it resides and
 105460 instantiate the batch job in another batch queue. A batch job is moved by a request to the batch
 105461 server that manages the batch job. The *qmove* utility is a user-accessible batch client that requests
 105462 the movement of one or more batch jobs.

105463 The *qmove* utility shall move those batch jobs, and only those batch jobs, for which a batch
 105464 *job_identifier* is presented to the utility.

105465 The *qmove* utility shall move batch jobs in the order in which the corresponding batch
 105466 *job_identifiers* are presented to the utility.

105467 If the *qmove* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 105468 process the remaining batch *job_identifiers*, if any.

105469 The *qmove* utility shall move batch jobs by sending a *Move Job Request* to the batch server that
 105470 manages each batch job. The *qmove* utility shall not exit before the batch jobs corresponding to all
 105471 successfully processed batch *job_identifiers* have been moved.

105472 **OPTIONS**

105473 None.

105474 **OPERANDS**

105475 The *qmove* utility shall accept one operand that conforms to the syntax for a destination (see
 105476 [Section 3.3.2](#), on page 2450).

105477 The *qmove* utility shall accept one or more operands that conform to the syntax for a batch
 105478 *job_identifier* (see [Section 3.3.1](#), on page 2449).

105479 **STDIN**

105480 Not used.

105481 **INPUT FILES**

105482 None.

105483 **ENVIRONMENT VARIABLES**105484 The following environment variables shall affect the execution of *qmove*:

105485 *LANG* Provide a default value for the internationalization variables that are unset or null.
 105486 (See [XBD Section 8.2](#) (on page 174) the precedence of internationalization variables
 105487 used to determine the values of locale categories.)

105488 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105489 internationalization variables.

105490 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105491 characters (for example, single-byte as opposed to multi-byte characters in
 105492 arguments).

105493 *LC_MESSAGES*

105494 Determine the locale that should be used to affect the format and contents of
 105495 diagnostic messages written to standard error.

- 105496 *LOGNAME* Determine the login name of the user.
- 105497 **ASYNCHRONOUS EVENTS**
- 105498 Default.
- 105499 **STDOUT**
- 105500 None.
- 105501 **STDERR**
- 105502 The standard error shall be used only for diagnostic messages.
- 105503 **OUTPUT FILES**
- 105504 None.
- 105505 **EXTENDED DESCRIPTION**
- 105506 None.
- 105507 **EXIT STATUS**
- 105508 The following exit values shall be returned:
- 105509 0 Successful completion.
- 105510 >0 An error occurred.
- 105511 **CONSEQUENCES OF ERRORS**
- 105512 In addition to the default behavior, the *qmove* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qmove* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.
- 105513
- 105514
- 105515
- 105516
- 105517 **APPLICATION USAGE**
- 105518 None.
- 105519 **EXAMPLES**
- 105520 None.
- 105521 **RATIONALE**
- 105522 The *qmove* utility allows users to move jobs between queues.
- 105523 The alternative to using the *qmove* utility—deleting the batch job and requeuing it—entails considerably more typing.
- 105524
- 105525 Since the means of selecting jobs based on attributes has been encapsulated in the *qselect* utility, the only option of the *qmove* utility concerns authorization. The **-u** option provides the user with the convenience of changing the user identifier under which the batch job will execute.
- 105526
- 105527
- 105528 Minimalism and consistency have taken precedence over convenience; the **-u** option has been deleted because the equivalent capability exists with the **-u** option of the *qalter* utility.
- 105529
- 105530 **FUTURE DIRECTIONS**
- 105531 The *qmove* utility may be removed in a future version.
- 105532 **SEE ALSO**
- 105533 [Chapter 3](#) (on page 2427), *qalter*, *qselect*
- 105534 [XBD Chapter 8](#) (on page 173)

105535 **CHANGE HISTORY**

105536 Derived from IEEE Std 1003.2d-1994.

105537 **Issue 6**

105538 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

105539 **Issue 7**

105540 The *qmove* utility is marked obsolescent.

105541 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105542 **NAME**

105543 qmsg ‡send message to batch jobs

105544 **SYNOPSIS**105545 OB BE `qmsg [-EO] message_string job_identifier...`105546 **DESCRIPTION**

105547 To send a message to a batch job is to request that a server write a message string into one or
 105548 more output files of the batch job. A message is sent to a batch job by a request to the batch
 105549 server that manages the batch job. The *qmsg* utility is a user-accessible batch client that requests
 105550 the sending of messages to one or more batch jobs.

105551 The *qmsg* utility shall write messages into the files of batch jobs by sending a *Job Message Request*
 105552 to the batch server that manages the batch job. The *qmsg* utility shall not directly write the
 105553 message into the files of the batch job.

105554 The *qmsg* utility shall send a *Job Message Request* for those batch jobs, and only those batch jobs,
 105555 for which a batch *job_identifier* is presented to the utility.

105556 The *qmsg* utility shall send *Job Message Requests* for batch jobs in the order in which their batch
 105557 *job_identifiers* are presented to the utility.

105558 If the *qmsg* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
 105559 process the remaining batch *job_identifiers*, if any.

105560 The *qmsg* utility shall not exit before a *Job Message Request* has been sent to the server that
 105561 manages the batch job that corresponds to each successfully processed batch *job_identifier*.

105562 **OPTIONS**

105563 The *qmsg* utility shall conform to XBD [Section 12.2](#) (on page 216).

105564 The following options shall be supported by the implementation:

105565 **-E** Specify that the message is written to the standard error of each batch job.

105566 The *qmsg* utility shall write the message into the standard error of the batch job.

105567 **-O** Specify that the message is written to the standard output of each batch job.

105568 The *qmsg* utility shall write the message into the standard output of the batch job.

105569 If neither the **-O** nor the **-E** option is presented to the *qmsg* utility, the utility shall write the
 105570 message into an implementation-defined file. The conformance document for the
 105571 implementation shall describe the name and location of the implementation-defined file. If both
 105572 the **-O** and the **-E** options are presented to the *qmsg* utility, then the utility shall write the
 105573 messages to both standard output and standard error.

105574 **OPERANDS**

105575 The *qmsg* utility shall accept a minimum of two operands, *message_string* and one or more batch
 105576 *job_identifiers*.

105577 The *message_string* operand shall be the string to be written to one or more output files of the
 105578 batch job followed by a <newline>. If the string contains <blank> characters, then the
 105579 application shall ensure that the string is quoted. The *message_string* shall be encoded in the
 105580 portable character set (see XBD [Section 6.1](#), on page 125).

105581 All remaining operands are batch *job_identifiers* that conform to the syntax for a batch
 105582 *job_identifier* (see [Section 3.3.1](#), on page 2449).

105583 **STDIN**

105584 Not used.

105585 **INPUT FILES**

105586 None.

105587 **ENVIRONMENT VARIABLES**105588 The following environment variables shall affect the execution of *qmsg*:

105589 *LANG* Provide a default value for the internationalization variables that are unset or null.
105590 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
105591 used to determine the values of locale categories.)

105592 *LC_ALL* If set to a non-empty string value, override the values of all the other
105593 internationalization variables.

105594 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
105595 characters (for example, single-byte as opposed to multi-byte characters in
105596 arguments).

105597 *LC_MESSAGES*

105598 Determine the locale that should be used to affect the format and contents of
105599 diagnostic messages written to standard error.

105600 *LOGNAME* Determine the login name of the user.

105601 **ASYNCHRONOUS EVENTS**

105602 Default.

105603 **STDOUT**

105604 None.

105605 **STDERR**

105606 The standard error shall be used only for diagnostic messages.

105607 **OUTPUT FILES**

105608 None.

105609 **EXTENDED DESCRIPTION**

105610 None.

105611 **EXIT STATUS**

105612 The following exit values shall be returned:

105613 0 Successful completion.

105614 >0 An error occurred.

105615 **CONSEQUENCES OF ERRORS**

105616 In addition to the default behavior, the *qmsg* utility shall not be required to write a diagnostic
105617 message to standard error when the error reply received from a batch server indicates that the
105618 batch *job_identifier* does not exist on the server. Whether or not the *qmsg* utility waits to output
105619 the diagnostic message while attempting to locate the job on other servers is implementation-
105620 defined.

105621 APPLICATION USAGE

105622 None.

105623 EXAMPLES

105624 None.

105625 RATIONALE

105626 The *qmsg* utility allows users to write messages into the output files of running jobs. Users,
105627 including operators and administrators, have a number of occasions when they want to place
105628 messages in the output files of a batch job. For example, if a disk that is being used by a batch job
105629 is showing errors, the operator might note this in the standard error stream of the batch job.

105630 The options of the *qmsg* utility provide users with the means of placing the message in the
105631 output stream of their choice. The default output stream for the message—if the user does not
105632 designate an output stream—is implementation-defined, since many implementations will
105633 provide, as an extension to this volume of POSIX.1-2017, a log file that shows the history of
105634 utility execution.

105635 If users wish to send a message to a set of jobs that meet a selection criteria, the *qselect* utility can
105636 be used to acquire the appropriate list of job identifiers.

105637 The **-E** option allows users to place the message in the standard error stream of the batch job.

105638 The **-O** option allows users to place the message in the standard output stream of the batch job.

105639 Historically, the *qmsg* utility is an existing practice in the offerings of one or more implementors
105640 of an NQS-derived batch system. The utility has been found to be useful enough that it deserves
105641 to be included in this volume of POSIX.1-2017.

105642 FUTURE DIRECTIONS

105643 The *qmsg* utility may be removed in a future version.

105644 SEE ALSO

105645 [Chapter 3](#) (on page 2427), *qselect*

105646 [XBD Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

105647 CHANGE HISTORY

105648 Derived from IEEE Std 1003.2d-1994.

105649 Issue 6

105650 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

105651 Issue 7

105652 The *qmsg* utility is marked obsolescent.

105653 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105654 **NAME**

105655 qrerun — rerun batch jobs

105656 **SYNOPSIS**105657 OB BE `qrerun job_identifier...`105658 **DESCRIPTION**

105659 To rerun a batch job is to terminate the session leader of the batch job, delete any associated
 105660 checkpoint files, and return the batch job to the batch queued state. A batch job is rerun by a
 105661 request to the batch server that manages the batch job. The *qrerun* utility is a user-accessible
 105662 batch client that requests the rerunning of one or more batch jobs.

105663 The *qrerun* utility shall rerun those batch jobs for which a batch *job_identifier* is presented to the
 105664 utility.

105665 The *qrerun* utility shall rerun batch jobs in the order in which their batch *job_identifiers* are
 105666 presented to the utility.

105667 If the *qrerun* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
 105668 process the remaining batch *job_identifiers*, if any.

105669 The *qrerun* utility shall rerun batch jobs by sending a *Rerun Job Request* to the batch server that
 105670 manages each batch job.

105671 For each successfully processed batch *job_identifier*, the *qrerun* utility shall have rerun the
 105672 corresponding batch job at the time the utility exits.

105673 **OPTIONS**

105674 None.

105675 **OPERANDS**

105676 The *qrerun* utility shall accept one or more operands that conform to the syntax for a batch
 105677 *job_identifier* (see [Section 3.3.1](#), on page 2449).

105678 **STDIN**

105679 Not used.

105680 **INPUT FILES**

105681 None.

105682 **ENVIRONMENT VARIABLES**105683 The following environment variables shall affect the execution of *qrerun*:

105684 *LANG* Provide a default value for the internationalization variables that are unset or null.
 105685 (See [XBD Section 8.2](#) (on page 174) the precedence of internationalization variables
 105686 used to determine the values of locale categories.)

105687 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105688 internationalization variables.

105689 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105690 characters (for example, single-byte as opposed to multi-byte characters in
 105691 arguments).

105692 *LC_MESSAGES*

105693 Determine the locale that should be used to affect the format and contents of
 105694 diagnostic messages written to standard error.

- 105695 *LOGNAME* Determine the login name of the user.
- 105696 **ASYNCHRONOUS EVENTS**
- 105697 Default.
- 105698 **STDOUT**
- 105699 None.
- 105700 **STDERR**
- 105701 The standard error shall be used only for diagnostic messages.
- 105702 **OUTPUT FILES**
- 105703 None.
- 105704 **EXTENDED DESCRIPTION**
- 105705 None.
- 105706 **EXIT STATUS**
- 105707 The following exit values shall be returned:
- 105708 0 Successful completion.
- 105709 >0 An error occurred.
- 105710 **CONSEQUENCES OF ERRORS**
- 105711 In addition to the default behavior, the *qrerun* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qrerun* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.
- 105712
- 105713
- 105714
- 105715
- 105716 **APPLICATION USAGE**
- 105717 None.
- 105718 **EXAMPLES**
- 105719 None.
- 105720 **RATIONALE**
- 105721 The *qrerun* utility allows users to cause jobs in the running state to exit and rerun.
- 105722 The *qrerun* utility is a new utility, *vis-a-vis* existing practice, that has been defined in this volume of POSIX.1-2017 to correct user-perceived deficiencies in the existing practice.
- 105723
- 105724 **FUTURE DIRECTIONS**
- 105725 The *qrerun* utility may be removed in a future version.
- 105726 **SEE ALSO**
- 105727 [Chapter 3](#) (on page 2427)
- 105728 [XBD Chapter 8](#) (on page 173)
- 105729 **CHANGE HISTORY**
- 105730 Derived from IEEE Std 1003.2d-1994.
- 105731 **Issue 6**
- 105732 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

105733 **Issue 7**

105734 The *qrerun* utility is marked obsolescent.

105735 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105736 **NAME**

105737 qrls — release batch jobs

105738 **SYNOPSIS**105739 OB BE `qrls [-h hold_list] job_identifier...`105740 **DESCRIPTION**

105741 A batch job might have one or more holds, which prevent the batch job from executing. A batch
 105742 job from which all the holds have been removed becomes eligible for execution and is said to
 105743 have been released. A batch job hold is removed by sending a request to the batch server that
 105744 manages the batch job. The *qrls* utility is a user-accessible client of batch services that requests
 105745 holds be removed from one or more batch jobs.

105746 The *qrls* utility shall remove one or more holds from those batch jobs for which a batch
 105747 *job_identifier* is presented to the utility.

105748 The *qrls* utility shall remove holds from batch jobs in the order in which their batch *job_identifiers*
 105749 are presented to the utility.

105750 If the *qrls* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 105751 process the remaining batch *job_identifiers*, if any.

105752 The *qrls* utility shall remove holds on each batch job by sending a *Release Job Request* to the batch
 105753 server that manages the batch job.

105754 The *qrls* utility shall not exit until the holds have been removed from the batch job
 105755 corresponding to each successfully processed batch *job_identifier*.

105756 **OPTIONS**

105757 The *qrls* utility shall conform to XBD [Section 12.2](#) (on page 216).

105758 The following option shall be supported by the implementation:

105759 **-h *hold_list*** Define the types of holds to be removed from the batch job.

105760 The *qrls* **-h** option shall accept a value for the *hold_list* option-argument that is a
 105761 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 105762 [6.1](#), on page 125).

105763 The *qrls* utility shall accept a value for the *hold_list* option-argument that is a string
 105764 of one or more of the characters 'u', 's', or 'o', or the single character 'n'.

105765 For each unique character in the *hold_list* option-argument, the *qrls* utility shall add
 105766 a value to the *Hold_Types* attribute of the batch job as follows, each representing a
 105767 different hold type:

105768 u USER

105769 s SYSTEM

105770 o OPERATOR

105771 If any of these characters are duplicated in the *hold_list* option-argument, the
 105772 duplicates shall be ignored.

105773 An existing *Hold_Types* attribute can be cleared by the following hold type:

105774 n NO_HOLD

105775 The *qrls* utility shall consider it an error if any hold type other than 'n' is
 105776 combined with hold type 'n'.

105777 Strictly conforming applications shall not repeat any of the characters 'u', 's',
 105778 'o', or 'n' within the *hold_list* option-argument. The *qrln* utility shall permit the
 105779 repetition of characters, but shall not assign additional meaning to the repeated
 105780 characters.

105781 An implementation may define other hold types. The conformance document for
 105782 an implementation shall describe any additional hold types, how they are
 105783 specified, their internal behavior, and how they affect the behavior of the utility.

105784 If the **-h** option is not presented to the *qrln* utility, the implementation shall remove
 105785 the USER hold in the *Hold_Types* attribute.

105786 OPERANDS

105787 The *qrln* utility shall accept one or more operands that conform to the syntax for a batch
 105788 *job_identifier* (see [Section 3.3.1](#), on page 2449).

105789 STDIN

105790 Not used.

105791 INPUT FILES

105792 None.

105793 ENVIRONMENT VARIABLES

105794 The following environment variables shall affect the execution of *qrln*:

105795 *LANG* Provide a default value for the internationalization variables that are unset or null.
 105796 (See [XBD Section 8.2](#) (on page 174) the precedence of internationalization variables
 105797 used to determine the values of locale categories.)

105798 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105799 internationalization variables.

105800 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105801 characters (for example, single-byte as opposed to multi-byte characters in
 105802 arguments).

105803 *LC_MESSAGES*

105804 Determine the locale that should be used to affect the format and contents of
 105805 diagnostic messages written to standard error.

105806 *LOGNAME* Determine the login name of the user.

105807 ASYNCHRONOUS EVENTS

105808 Default.

105809 STDOUT

105810 None.

105811 STDERR

105812 The standard error shall be used only for diagnostic messages.

105813 OUTPUT FILES

105814 None.

105815 EXTENDED DESCRIPTION

105816 None.

105817 EXIT STATUS

105818 The following exit values shall be returned:

105819 0 Successful completion.

105820 >0 An error occurred.

105821 CONSEQUENCES OF ERRORS

105822 In addition to the default behavior, the *qrln* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qrln* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

105827 APPLICATION USAGE

105828 None.

105829 EXAMPLES

105830 None.

105831 RATIONALE

105832 The *qrln* utility allows users, operators, and administrators to remove holds from jobs.

105833 The *qrln* utility does not support any job selection options or wildcard arguments. Users may acquire a list of jobs selected by attributes using the *qselect* utility. For example, a user could select all of their held jobs.

105836 The **-h** option allows the user to specify the type of hold that is to be removed. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The batch server that manages the batch job will verify whether the user is authorized to remove the specified hold for the batch job. If more than one type of hold has been placed on the batch job, a user may wish to remove only some of them.

105841 Mail is not required on release because the administrator has the tools and libraries to build this option if required.

105843 The *qrln* utility is a new utility *vis-a-vis* existing practice; it has been defined in this volume of POSIX.1-2017 as the natural complement to the *qhold* utility.

105845 FUTURE DIRECTIONS

105846 The *qrln* utility may be removed in a future version.

105847 SEE ALSO

105848 [Chapter 3](#) (on page 2427), *qhold*, *qselect*

105849 [XBD Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

105850 CHANGE HISTORY

105851 Derived from IEEE Std 1003.2d-1994.

105852 Issue 6

105853 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

105854 Issue 7

105855 The *qrln* utility is marked obsolescent.

105856 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105857 **NAME**

105858 qselect ‡select batch jobs

105859 **SYNOPSIS**

```
105860 OB BE qselect [-a [op]date_time] [-A account_string] [-c [op]interval]
105861 [-h hold_list] [-l resource_list] [-N name] [-p [op]priority]
105862 [-q destination] [-r y|n] [-s states] [-u user_list]
```

105863 **DESCRIPTION**

105864 To select a set of batch jobs is to return the batch *job_identifiers* for each batch job that meets a list
 105865 of selection criteria. A set of batch jobs is selected by a request to a batch server. The *qselect* utility
 105866 is a user-accessible batch client that requests the selection of batch jobs.

105867 Upon successful completion, the *qselect* utility shall have returned a list of zero or more batch
 105868 *job_identifiers* that meet the criteria specified by the options and option-arguments presented to
 105869 the utility.

105870 The *qselect* utility shall select batch jobs by sending a *Select Jobs Request* to a batch server. The
 105871 *qselect* utility shall not exit until the server replies to each request generated.

105872 For each option presented to the *qselect* utility, the utility shall restrict the set of selected batch
 105873 jobs as described in the OPTIONS section.

105874 The *qselect* utility shall not restrict selection of batch jobs except by authorization and as required
 105875 by the options presented to the utility.

105876 When an option is specified with a mandatory or optional *op* component to the option-
 105877 argument, then *op* shall specify a relation between the value of a certain batch job attribute and
 105878 the *value* component of the option-argument. If an *op* is allowable on an option, then the
 105879 description of the option letter indicates the *op* as either mandatory or optional. Acceptable
 105880 strings for the *op* component, and the relation the string indicates, are shown in the following
 105881 list:

105882 .eq. The value represented by the attribute of the batch job is equal to the value represented
 105883 by the option-argument.

105884 .ge. The value represented by the attribute of the batch job is greater than or equal to the
 105885 value represented by the option-argument.

105886 .gt. The value represented by the attribute of the batch job is greater than the value
 105887 represented by the option-argument.

105888 .lt. The value represented by the attribute of the batch job is less than the value represented
 105889 by the option-argument.

105890 .le. The value represented by the attribute of the batch job is less than or equal to the value
 105891 represented by the option-argument.

105892 .ne. The value represented by the attribute of the batch job is not equal to the value
 105893 represented by the option-argument.

105894 **OPTIONS**

105895 The *qselect* utility shall conform to XBD [Section 12.2](#) (on page 216).

105896 The following options shall be supported by the implementation:

105897 **-a** [op]date_time

105898 Restrict selection to a specific time, or a range of times.

105899 The *qselect* utility shall select only batch jobs for which the value of the

105900 *Execution_Time* attribute is related to the Epoch equivalent of the local time
 105901 expressed by the value of the *date_time* component of the option-argument in the
 105902 manner indicated by the value of the *op* component of the option-argument.

105903 The *qselect* utility shall accept a *date_time* component of the option-argument that
 105904 conforms to the syntax of the *time* operand of the *touch* utility.

105905 If the *op* component of the option-argument is not presented to the *qselect* utility,
 105906 the utility shall select batch jobs for which the *Execution_Time* attribute is equal to
 105907 the *date_time* component of the option-argument.

105908 When comparing times, the *qselect* utility shall use the following definitions for the
 105909 *op* component of the option-argument:

105910 *.eq.* The time represented by value of the *Execution_Time* attribute of the batch
 105911 job is equal to the time represented by the *date_time* component of the
 105912 option-argument.

105913 *.ge.* The time represented by value of the *Execution_Time* attribute of the batch
 105914 job is after or equal to the time represented by the *date_time* component of
 105915 the option-argument.

105916 *.gt.* The time represented by value of the *Execution_Time* attribute of the batch
 105917 job is after the time represented by the *date_time* component of the option-
 105918 argument.

105919 *.lt.* The time represented by value of the *Execution_Time* attribute of the batch
 105920 job is before the time represented by the *date_time* component of the
 105921 option-argument.

105922 *.le.* The time represented by value of the *Execution_Time* attribute of the batch
 105923 job is before or equal to the time represented by the *date_time* component
 105924 of the option-argument.

105925 *.ne.* The time represented by value of the *Execution_Time* attribute of the batch
 105926 job is not equal to the time represented by the *date_time* component of the
 105927 option-argument.

105928 The *qselect* utility shall accept the defined character strings for the *op* component of
 105929 the option-argument.

105930 **-A** *account_string*

105931 Restrict selection to the batch jobs charging a specified account.

105932 The *qselect* utility shall select only batch jobs for which the value of the
 105933 *Account_Name* attribute of the batch job matches the value of the *account_string*
 105934 option-argument.

105935 The syntax of the *account_string* option-argument is unspecified.

105936 **-c** [*op*]*interval*

105937 Restrict selection to batch jobs within a range of checkpoint intervals.

105938 The *qselect* utility shall select only batch jobs for which the value of the *Checkpoint*
 105939 attribute relates to the value of the *interval* component of the option-argument in
 105940 the manner indicated by the value of the *op* component of the option-argument.

105941 If the *op* component of the option-argument is omitted, the *qselect* utility shall select
 105942 batch jobs for which the value of the *Checkpoint* attribute is equal to the value of the
 105943 *interval* component of the option-argument.

105944 When comparing checkpoint intervals, the *qselect* utility shall use the following
 105945 definitions for the *op* component of the option-argument:

105946 *.eq.* The value of the *Checkpoint* attribute of the batch job equals the value of
 105947 the *interval* component of the option-argument.

105948 *.ge.* The value of the *Checkpoint* attribute of the batch job is greater than or
 105949 equal to the value of the *interval* component option-argument.

105950 *.gt.* The value of the *Checkpoint* attribute of the batch job is greater than the
 105951 value of the *interval* component option-argument.

105952 *.lt.* The value of the *Checkpoint* attribute of the batch job is less than the value
 105953 of the *interval* component option-argument.

105954 *.le.* The value of the *Checkpoint* attribute of the batch job is less than or equal
 105955 to the value of the *interval* component option-argument.

105956 *.ne.* The value of the *Checkpoint* attribute of the batch job does not equal the
 105957 value of the *interval* component option-argument.

105958 The *qselect* utility shall accept the defined character strings for the *op* component of
 105959 the option-argument.

105960 The ordering relationship for the values of the interval option-argument is defined
 105961 to be:

105962 ``n' .gt. `s' .gt. `c=minutes' .ge. `c'`

105963 When comparing *Checkpoint* attributes with an interval having the value of the
 105964 single character 'u', only equality or inequality are valid comparisons.

105965 **-h hold_list** Restrict selection to batch jobs that have a specific type of hold.

105966 The *qselect* utility shall select only batch jobs for which the value of the *Hold_Types*
 105967 attribute matches the value of the *hold_list* option-argument.

105968 The *qselect* **-h** option shall accept a value for the *hold_list* option-argument that is a
 105969 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 105970 [6.1](#), on page 125).

105971 The *qselect* utility shall accept a value for the *hold_list* option-argument that is a
 105972 string of one or more of the characters 'u', 's', or 'o', or the single character
 105973 'n'.

105974 Each unique character in the *hold_list* option-argument of the *qselect* utility is
 105975 defined as follows, each representing a different hold type:

105976 u USER

105977 s SYSTEM

105978 o OPERATOR

105979 If any of these characters are duplicated in the *hold_list* option-argument, the
 105980 duplicates shall be ignored.

105981 The *qselect* utility shall consider it an error if any hold type other than 'n' is
 105982 combined with hold type 'n'.

105983 Strictly conforming applications shall not repeat any of the characters 'u', 's',
 105984 'o', or 'n' within the *hold_list* option-argument. The *qselect* utility shall permit

105985 the repetition of characters, but shall not assign additional meaning to the repeated
105986 characters.

105987 An implementation may define other hold types. The conformance document for
105988 an implementation shall describe any additional hold types, how they are
105989 specified, their internal behavior, and how they affect the behavior of the utility.

105990 **-I *resource_list***

105991 Restrict selection to batch jobs with specified resource limits and attributes.

105992 The *qselect* utility shall accept a *resource_list* option-argument with the following
105993 syntax:

105994 *resource_name op value [, , resource_name op value , , ...]*

105995 When comparing resource values, the *qselect* utility shall use the following
105996 definitions for the *op* component of the option-argument:

105997 *.eq.* The value of the resource of the same name in the *Resource_List* attribute
105998 of the batch job equals the value of the value component of the option-
105999 argument.

106000 *.ge.* The value of the resource of the same name in the *Resource_List* attribute
106001 of the batch job is greater than or equal to the value of the *value*
106002 component of the option-argument.

106003 *.gt.* The value of the resource of the same name in the *Resource_List* attribute
106004 of the batch job is greater than the value of the value component of the
106005 option-argument.

106006 *.lt.* The value of the resource of the same name in the *Resource_List* attribute
106007 of the batch job is less than the value of the value component of the
106008 option-argument.

106009 *.ne.* The value of the resource of the same name in the *Resource_List* attribute
106010 of the batch job does not equal the value of the value component of the
106011 option-argument.

106012 *.le.* The value of the resource of the same name in the *Resource_List* attribute
106013 of the batch job is less than or equal to the value of the *value* component of
106014 the option-argument.

106015 When comparing the limit of a *Resource_List* attribute with the *value* component of
106016 the option-argument, if the limit, the value, or both are non-numeric, only equality
106017 or inequality are valid comparisons.

106018 The *qselect* utility shall select only batch jobs for which the values of the
106019 *resource_names* listed in the *resource_list* option-argument match the corresponding
106020 limits of the *Resource_List* attribute of the batch job.

106021 Limits of *resource_names* present in the *Resource_List* attribute of the batch job that
106022 have no corresponding values in the *resource_list* option-argument shall not be
106023 considered when selecting batch jobs.

106024 **-N *name*** Restrict selection to batch jobs with a specified name.

106025 The *qselect* utility shall select only batch jobs for which the value of the *Job_Name*
106026 attribute matches the value of the *name* option-argument. The string specified in
106027 the *name* option-argument shall be passed, uninterpreted, to the server. This allows
106028 an implementation to match ``wildcard'' patterns against batch job names.

106029 An implementation shall describe in the conformance document the format it
106030 supports for matching against the *Job_Name* attribute.

106031 **-p** [*op*]*priority*

106032 Restrict selection to batch jobs of the specified priority or range of priorities.

106033 The *qselect* utility shall select only batch jobs for which the value of the *Priority*
106034 attribute of the batch job relates to the value of the *priority* component of the
106035 option-argument in the manner indicated by the value of the *op* component of the
106036 option-argument.

106037 If the *op* component of the option-argument is omitted, the *qselect* utility shall select
106038 batch jobs for which the value of the *Priority* attribute of the batch job is equal to
106039 the value of the *priority* component of the option-argument.

106040 When comparing priority values, the *qselect* utility shall use the following
106041 definitions for the *op* component of the option-argument:

106042 .eq. The value of the *Priority* attribute of the batch job equals the value of the
106043 *priority* component of the option-argument.

106044 .ge. The value of the *Priority* attribute of the batch job is greater than or equal
106045 to the value of the *priority* component option-argument.

106046 .gt. The value of the *Priority* attribute of the batch job is greater than the value
106047 of the *priority* component option-argument.

106048 .lt. The value of the *Priority* attribute of the batch job is less than the value of
106049 the *priority* component option-argument.

106050 .lte. The value of the *Priority* attribute of the batch job is less than or equal to
106051 the value of the *priority* component option-argument.

106052 .ne. The value of the *Priority* attribute of the batch job does not equal the value
106053 of the *priority* component option-argument.

106054 **-q** *destination*

106055 Restrict selection to the specified batch queue or server, or both.

106056 The *qselect* utility shall select only batch jobs that are located at the destination
106057 indicated by the value of the *destination* option-argument.

106058 The destination defines a batch queue, a server, or a batch queue at a server.

106059 The *qselect* utility shall accept an option-argument for the **-q** option that conforms
106060 to the syntax for a destination. If the **-q** option is not presented to the *qselect* utility,
106061 the utility shall select batch jobs from all batch queues at the default batch server.

106062 If the option-argument describes only a batch queue, the *qselect* utility shall select
106063 only batch jobs from the batch queue of the specified name at the default batch
106064 server. The means by which *qselect* determines the default server is
106065 implementation-defined.

106066 If the option-argument describes only a batch server, the *qselect* utility shall select
106067 batch jobs from all the batch queues at that batch server.

106068 If the option-argument describes both a batch queue and a batch server, the *qselect*
106069 utility shall select only batch jobs from the specified batch queue at the specified
106070 server.

- 106071 **-r y | n** Restrict selection to batch jobs with the specified rerunability status.
- 106072 The *qselect* utility shall select only batch jobs for which the value of the *Rerunable*
- 106073 attribute of the batch job matches the value of the option-argument.
- 106074 The *qselect* utility shall accept a value for the option-argument that consists of
- 106075 either the single character 'y' or the single character 'n'. The character 'y'
- 106076 represents the value TRUE, and the character 'n' represents the value FALSE.
- 106077 **-s states** Restrict selection to batch jobs in the specified states.
- 106078 The *qselect* utility shall accept an option-argument that consists of any combination
- 106079 of the characters 'e', 'q', 'r', 'w', 'h', and 't'.
- 106080 Conforming applications shall not repeat any character in the option-argument.
- 106081 The *qselect* utility shall permit the repetition of characters in the option-argument,
- 106082 but shall not assign additional meaning to repeated characters.
- 106083 The *qselect* utility shall interpret the characters in the *states* option-argument as
- 106084 follows:
- 106085 e Represents the EXITING state.
- 106086 q Represents the QUEUED state.
- 106087 r Represents the RUNNING state.
- 106088 t Represents the TRANSITING state.
- 106089 h Represents the HELD state.
- 106090 w Represents the WAITING state.
- 106091 For each character in the *states* option-argument, the *qselect* utility shall select batch
- 106092 jobs in the corresponding state.
- 106093 **-u user_list** Restrict selection to batch jobs owned by the specified user names.
- 106094 The *qselect* utility shall select only the batch jobs of those users specified in the
- 106095 *user_list* option-argument.
- 106096 The *qselect* utility shall accept a *user_list* option-argument that conforms to the
- 106097 following syntax:
- 106098 *username[@host] [, , username[@host] , , ...]*
- 106099 The *qselect* utility shall accept only one user name that is missing a corresponding
- 106100 host name. The *qselect* utility shall accept only one user name per named host.
- 106101 **OPERANDS**
- 106102 None.
- 106103 **STDIN**
- 106104 Not used.
- 106105 **INPUT FILES**
- 106106 None.
- 106107 **ENVIRONMENT VARIABLES**
- 106108 The following environment variables shall affect the execution of *qselect*:

- 106109 *LANG* Provide a default value for the internationalization variables that are unset or null.
 106110 (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables
 106111 used to determine the values of locale categories.)
- 106112 *LC_ALL* If set to a non-empty string value, override the values of all the other
 106113 internationalization variables.
- 106114 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 106115 characters (for example, single-byte as opposed to multi-byte characters in
 106116 arguments).
- 106117 *LC_MESSAGES*
 106118 Determine the locale that should be used to affect the format and contents of
 106119 diagnostic messages written to standard error.
- 106120 *LOGNAME* Determine the login name of the user.
- 106121 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
 106122 unset or null, an unspecified default timezone shall be used.
- 106123 **ASYNCHRONOUS EVENTS**
- 106124 Default.
- 106125 **STDOUT**
- 106126 The *qselect* utility shall write zero or more batch *job_identifiers* to standard output.
- 106127 The *qselect* utility shall separate the batch *job_identifiers* written to standard output by white
 106128 space.
- 106129 The *qselect* utility shall write batch *job_identifiers* in the following format:
- 106130 *sequence_number.server_name@server*
- 106131 **STDERR**
- 106132 The standard error shall be used only for diagnostic messages.
- 106133 **OUTPUT FILES**
- 106134 None.
- 106135 **EXTENDED DESCRIPTION**
- 106136 None.
- 106137 **EXIT STATUS**
- 106138 The following exit values shall be returned:
- 106139 0 Successful completion.
- 106140 >0 An error occurred.
- 106141 **CONSEQUENCES OF ERRORS**
- 106142 Default.

106143 **APPLICATION USAGE**

106144 None.

106145 **EXAMPLES**

106146 The following example shows how a user might use the *qselect* utility in conjunction with the
 106147 *qdel* utility to delete all of his or her jobs in the queued state without affecting any jobs that are
 106148 already running:

106149 `qdel $(qselect -s q)`

106150 or:

106151 `qselect -s q || xargs qdel`106152 **RATIONALE**

106153 The *qselect* utility allows users to acquire a list of job identifiers that match user-specified
 106154 selection criteria. The list of identifiers returned by the *qselect* utility conforms to the syntax of
 106155 the batch job identifier list processed by a utility such as *qmove*, *qdel*, and *qrls*. The *qselect* utility
 106156 is thus a powerful tool for causing another batch system utility to act upon a set of jobs that
 106157 match a list of selection criteria.

106158 The options of the *qselect* utility let the user apply a number of useful filters for selecting jobs.
 106159 Each option further restricts the selection of jobs. Many of the selection options allow the
 106160 specification of a relational operator. The FORTRAN-like syntax of the operator—that is,
 106161 ".lt." ¶was chosen rather than the C-like "<=" meta-characters.

106162 The **-a** option allows users to restrict the selected jobs to those that have been submitted (or
 106163 altered) to wait until a particular time. The time period is determined by the argument of this
 106164 option, which includes both a time and an operator ¶it is thus possible to select jobs waiting
 106165 until a specific time, jobs waiting until after a certain time, or those waiting for a time before the
 106166 specified time.

106167 The **-A** option allows users to restrict the selected jobs to those that have been submitted (or
 106168 altered) to charge a particular account.

106169 The **-c** option allows users to restrict the selected jobs to those whose checkpointing interval
 106170 falls within the specified range.

106171 The **-l** option allows users to select those jobs whose resource limits fall within the range
 106172 indicated by the value of the option. For example, a user could select those jobs for which the
 106173 CPU time limit is greater than two hours.

106174 The **-N** option allows users to select jobs by job name. For instance, all the parts of a task that
 106175 have been divided in parallel jobs might be given the same name, and thus manipulated as a
 106176 group by means of this option.

106177 The **-q** option allows users to select jobs in a specified queue.

106178 The **-r** option allows users to select only those jobs with a specified rerun criteria. For instance, a
 106179 user might select only those jobs that can be rerun for use with the *qrerun* utility.

106180 The **-s** option allows users to select only those jobs that are in a certain state.

106181 The **-u** option allows users to select jobs that have been submitted to execute under a particular
 106182 account.

106183 The selection criteria provided by the options of the *qselect* utility allow users to select jobs based
 106184 on all the appropriate attributes that can be assigned to jobs by the *qsub* utility.

106185 Historically, the *qselect* utility has not been a part of existing practice; it is an improvement that

106186 has been introduced in this volume of POSIX.1-2017.

106187 **FUTURE DIRECTIONS**

106188 The *qselect* utility may be removed in a future version.

106189 **SEE ALSO**

106190 [Chapter 3](#) (on page 2427), [qdel](#), [qrerun](#), [qrls](#), [qselect](#), [qsub](#), [touch](#)

106191 XBD [Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

106192 **CHANGE HISTORY**

106193 Derived from IEEE Std 1003.2d-1994.

106194 **Issue 7**

106195 The *qselect* utility is marked obsolescent.

106196 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

106197 **NAME**

106198 qsig ‡signal batch jobs

106199 **SYNOPSIS**106200 OB BE `qsig [-s signal] job_identifier...`106201 **DESCRIPTION**

106202 To signal a batch job is to send a signal to the session leader of the batch job. A batch job is
 106203 signaled by sending a request to the batch server that manages the batch job. The *qsig* utility is a
 106204 user-accessible batch client that requests the signaling of a batch job.

106205 The *qsig* utility shall signal those batch jobs for which a batch *job_identifier* is presented to the
 106206 utility. The *qsig* utility shall not signal any batch jobs whose batch *job_identifiers* are not
 106207 presented to the utility.

106208 The *qsig* utility shall signal batch jobs in the order in which the corresponding batch
 106209 *job_identifiers* are presented to the utility. If the *qsig* utility fails to process a batch *job_identifier*
 106210 successfully, the utility shall proceed to process the remaining batch *job_identifiers*, if any.

106211 The *qsig* utility shall signal batch jobs by sending a *Signal Job Request* to the batch server that
 106212 manages the batch job.

106213 For each successfully processed batch *job_identifier*, the *qsig* utility shall have received a
 106214 completion reply to each *Signal Job Request* sent to a batch server at the time the utility exits.

106215 **OPTIONS**106216 The *qsig* utility shall conform to XBD [Section 12.2](#) (on page 216).

106217 The following option shall be supported by the implementation:

106218 `-s signal` Define the signal to be sent to the batch job.

106219 The *qsig* utility shall accept a *signal* option-argument that is either a symbolic signal
 106220 name or an unsigned integer signal number (see the POSIX.1-1990 standard,
 106221 Section 3.3.1.1). The *qsig* utility shall accept signal names for which the SIG prefix
 106222 has been omitted.

106223 If the *signal* option-argument is a signal name, the *qsig* utility shall send that name.

106224 If the *signal* option-argument is a number, the *qsig* utility shall send the signal
 106225 value represented by the number.

106226 If the `-s` option is not presented to the *qsig* utility, the utility shall send the signal
 106227 SIGTERM to each signaled batch job.

106228 **OPERANDS**

106229 The *qsig* utility shall accept one or more operands that conform to the syntax for a batch
 106230 *job_identifier* (see [Section 3.3.1](#), on page 2449).

106231 **STDIN**

106232 Not used.

106233 **INPUT FILES**

106234 None.

106235 **ENVIRONMENT VARIABLES**106236 The following environment variables shall affect the execution of *qsig*:

- 106237 *LANG* Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 174) the precedence of internationalization variables used to determine the values of locale categories.)
- 106238
- 106239
- 106240 *LC_ALL* If set to a non-empty string value, override the values of all the other internationalization variables.
- 106241
- 106242 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 106243
- 106244
- 106245 *LC_MESSAGES*
- 106246 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
- 106247
- 106248 *LOGNAME* Determine the login name of the user.
- 106249 **ASYNCHRONOUS EVENTS**
- 106250 Default.
- 106251 **STDOUT**
- 106252 An implementation of the *qsig* utility may write informative messages to standard output.
- 106253 **STDERR**
- 106254 The standard error shall be used only for diagnostic messages.
- 106255 **OUTPUT FILES**
- 106256 None.
- 106257 **EXTENDED DESCRIPTION**
- 106258 None.
- 106259 **EXIT STATUS**
- 106260 The following exit values shall be returned:
- 106261 0 Successful completion.
- 106262 >0 An error occurred.
- 106263 **CONSEQUENCES OF ERRORS**
- 106264 In addition to the default behavior, the *qsig* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qsig* utility waits to output the diagnostic message while attempting to locate the batch job on other servers is implementation-defined.
- 106265
- 106266
- 106267
- 106268
- 106269 **APPLICATION USAGE**
- 106270 None.
- 106271 **EXAMPLES**
- 106272 None.
- 106273 **RATIONALE**
- 106274 The *qsig* utility allows users to signal batch jobs.
- 106275 A user may be unable to signal a batch job with the *kill* utility of the operating system for a number of reasons. First, the process ID of the batch job may be unknown to the user. Second, the processes of the batch job may be on a remote node. However, by virtue of communication between batch nodes, the *qsig* utility can arrange for the signaling of a process.
- 106276
- 106277
- 106278
- 106279 Because a batch job that is not running cannot be signaled, and because the signal may not

- 106280 terminate the batch job, the *qsig* utility is not a substitute for the *qdel* utility.
- 106281 The options of the *qsig* utility allow the user to specify the signal that is to be sent to the batch
106282 job.
- 106283 The `-s` option allows users to specify a signal by name or by number, and thus override the
106284 default signal. The POSIX.1-1990 standard defines signals by both name and number.
- 106285 The *qsig* utility is a new utility, *vis-a-vis* existing practice; it has been defined in this volume of
106286 POSIX.1-2017 in response to user-perceived shortcomings in existing practice.
- 106287 **FUTURE DIRECTIONS**
- 106288 The *qsig* utility may be removed in a future version.
- 106289 **SEE ALSO**
- 106290 [Chapter 3](#) (on page 2427), *kill*, *qdel*
- 106291 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 106292 **CHANGE HISTORY**
- 106293 Derived from IEEE Std 1003.2d-1994.
- 106294 **Issue 6**
- 106295 The `LC_TIME` and `TZ` entries are removed from the ENVIRONMENT VARIABLES section.
- 106296 **Issue 7**
- 106297 The *qsig* utility is marked obsolescent.
- 106298 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

106299 **NAME**

106300 qstat ‡show status of batch jobs

106301 **SYNOPSIS**106302 OB BE qstat [-f] *job_identifier...*106303 qstat -Q [-f] *destination...*106304 qstat -B [-f] *server_name...*106305 **DESCRIPTION**

106306 The status of a batch job, batch queue, or batch server is obtained by a request to the server. The
 106307 *qstat* utility is a user-accessible batch client that requests the status of one or more batch jobs,
 106308 batch queues, or servers, and writes the status information to standard output.

106309 For each successfully processed batch *job_identifier*, the *qstat* utility shall display information
 106310 about the corresponding batch job.

106311 For each successfully processed destination, the *qstat* utility shall display information about the
 106312 corresponding batch queue.

106313 For each successfully processed server name, the *qstat* utility shall display information about the
 106314 corresponding server.

106315 The *qstat* utility shall acquire batch job status information by sending a *Job Status Request* to a
 106316 batch server. The *qstat* utility shall acquire batch queue status information by sending a *Queue*
 106317 *Status Request* to a batch server. The *qstat* utility shall acquire server status information by
 106318 sending a *Server Status Request* to a batch server.

106319 **OPTIONS**106320 The *qstat* utility shall conform to XBD [Section 12.2](#) (on page 216).

106321 The following options shall be supported by the implementation:

106322 **-f** Specify that a full display is produced.

106323 The minimum contents of a full display are specified in the STDOUT section.

106324 Additional contents and format of a full display are implementation-defined.

106325 **-Q** Specify that the operand is a destination.106326 The *qstat* utility shall display information about each batch queue at each
 106327 destination identified as an operand.106328 **-B** Specify that the operand is a server name.106329 The *qstat* utility shall display information about each server identified as an
 106330 operand.106331 **OPERANDS**106332 If the **-Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that
 106333 conform to the syntax for a destination (see [Section 3.3.2](#), on page 2450).106334 If the **-B** option is presented to the *qstat* utility, the utility shall accept one or more *server_name*
 106335 operands.106336 If neither the **-B** nor the **-Q** option is presented to the *qstat* utility, the utility shall accept one or
 106337 more operands that conform to the syntax for a batch *job_identifier* (see [Section 3.3.1](#), on page
 106338 2449).

- 106339 **STDIN**
 106340 Not used.
- 106341 **INPUT FILES**
 106342 None.
- 106343 **ENVIRONMENT VARIABLES**
 106344 The following environment variables shall affect the execution of *qstat*:
- 106345 *HOME* Determine the pathname of the user's home directory.
- 106346 *LANG* Provide a default value for the internationalization variables that are unset or null.
 106347 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
 106348 used to determine the values of locale categories.)
- 106349 *LC_ALL* If set to a non-empty string value, override the values of all the other
 106350 internationalization variables.
- 106351 *LC_COLLATE*
 106352 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 106353 character collating elements within regular expressions.
- 106354 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 106355 characters (for example, single-byte as opposed to multi-byte characters in
 106356 arguments).
- 106357 *LC_MESSAGES*
 106358 Determine the locale that should be used to affect the format and contents of
 106359 diagnostic messages written to standard error.
- 106360 *LC_NUMERIC*
 106361 Determine the locale for selecting the radix character used when writing floating-
 106362 point formatted output.
- 106363 **ASYNCHRONOUS EVENTS**
 106364 Default.
- 106365 **STDOUT**
 106366 If an operand presented to the *qstat* utility is a batch *job_identifier* and the *-f* option is not
 106367 specified, the *qstat* utility shall display the following items on a single line, in the stated order,
 106368 with white space between each item, for each successfully processed operand:
- 106369 The batch *job_identifier*
- 106370 The batch job name
- 106371 The *Job_Owner* attribute
- 106372 The CPU time used by the batch job
- 106373 The batch job state
- 106374 The batch job location
- 106375 If an operand presented to the *qstat* utility is a batch *job_identifier* and the *-f* option is specified,
 106376 the *qstat* utility shall display the following items for each success fully processed operand:
- 106377 The batch *job_identifier*
- 106378 The batch job name

106379	The <i>Job_Owner</i> attribute	
106380	The execution user ID	
106381	The CPU time used by the batch job	
106382	The batch job state	
106383	The batch job location	
106384	Additional implementation-defined information, if any, about the batch job or batch queue	
106385	If an operand presented to the <i>qstat</i> utility is a destination, the -Q option is specified, and the -f option is not specified, the <i>qstat</i> utility shall display the following items on a single line, in the stated order, with white space between each item, for each successfully processed operand:	
106386		
106387		
106388	The batch queue name	
106389	The maximum number of batch jobs that shall be run in the batch queue concurrently	
106390	The total number of batch jobs in the batch queue	
106391	The status of the batch queue	
106392	For each state, the number of batch jobs in that state in the batch queue and the name of the state	
106393		
106394	The type of batch queue (execution or routing)	
106395	If the operands presented to the <i>qstat</i> utility are destinations, the -Q option is specified, and the -f option is specified, the <i>qstat</i> utility shall display the following items for each successfully processed operand:	
106396		
106397		
106398	The batch queue name	
106399	The maximum number of batch jobs that shall be run in the batch queue concurrently	
106400	The total number of batch jobs in the batch queue	
106401	The status of the batch queue	
106402	For each state, the number of batch jobs in that state in the batch queue and the name of the state	
106403		
106404	The type of batch queue (execution or routing)	
106405	Additional implementation-defined information, if any, about the batch queue	
106406	If the operands presented to the <i>qstat</i> utility are batch server names, the -B option is specified, and the -f option is not specified, the <i>qstat</i> utility shall display the following items on a single line, in the stated order, with white space between each item, for each successfully processed operand:	
106407		
106408		
106409		
106410	The batch server name	
106411	The maximum number of batch jobs that shall be run in the batch queue concurrently	
106412	The total number of batch jobs managed by the batch server	
106413	The status of the batch server	
106414	For each state, the number of batch jobs in that state and the name of the state	
106415	If the operands presented to the <i>qstat</i> utility are server names, the -B option is specified, and the -f option is specified, the <i>qstat</i> utility shall display the following items for each successfully	
106416		

106417 processed operand:

106418 The server name

106419 The maximum number of batch jobs that shall be run in the batch queue concurrently

106420 The total number of batch jobs managed by the server

106421 The status of the server

106422 For each state, the number of batch jobs in that state and the name of the state

106423 Additional implementation-defined information, if any, about the server

106424 **STDERR**

106425 The standard error shall be used only for diagnostic messages.

106426 **OUTPUT FILES**

106427 None.

106428 **EXTENDED DESCRIPTION**

106429 None.

106430 **EXIT STATUS**

106431 The following exit values shall be returned:

106432 0 Successful completion.

106433 >0 An error occurred.

106434 **CONSEQUENCES OF ERRORS**

106435 In addition to the default behavior, the *qstat* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qstat* utility waits to output the diagnostic message while attempting to locate the batch job on other servers is implementation-defined.

106436

106437

106438

106439

106440 **APPLICATION USAGE**

106441 None.

106442 **EXAMPLES**

106443 None.

106444 **RATIONALE**

106445 The *qstat* utility allows users to display the status of jobs and list the batch jobs in queues.

106446 The operands of the *qstat* utility may be either job identifiers, queues (specified as destination identifiers), or batch server names. The **-Q** and **-B** options, or absence thereof, indicate the nature of the operands.

106447

106448

106449 The other options of the *qstat* utility allow the user to control the amount of information displayed and the format in which it is displayed. Should a user wish to display the status of a set of jobs that match a selection criteria, the *qselect* utility may be used to acquire such a list.

106450

106451

106452 The **-f** option allows users to request a “full” display in an implementation-defined format.

106453 Historically, the *qstat* utility has been a part of the NQS and its derivatives, the existing practice on which it is based.

106454

106455 **FUTURE DIRECTIONS**

106456 The *qstat* utility may be removed in a future version.

106457 **SEE ALSO**

106458 [Chapter 3](#) (on page 2427), *qselect*

106459 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

106460 **CHANGE HISTORY**

106461 Derived from IEEE Std 1003.2d-1994.

106462 **Issue 6**

106463 IEEE PASC Interpretation 1003.2 #191 is applied, removing the following ENVIRONMENT
106464 VARIABLES listed as affecting *qstat*: *COLUMNS*, *LINES*, *LOGNAME*, *TERM*, and *TZ*.

106465 The *LC_TIME* entry is also removed from the ENVIRONMENT VARIABLES section.

106466 **Issue 7**

106467 The *qstat* utility is marked obsolescent.

106468 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

106469 **NAME**

106470 qsub ‡submit a script

106471 **SYNOPSIS**

```

106472 OB BE qsub [-a date_time] [-A account_string] [-c interval]
106473 [-C directive_prefix] [-e path_name] [-h] [-j join_list]
106474 [-k keep_list] [-m mail_options] [-M mail_list] [-N name]
106475 [-o path_name] [-p priority] [-q destination] [-r y|n]
106476 [-S path_name_list] [-u user_list] [-v variable_list] [-V]
106477 [-z] [script]

```

106478 **DESCRIPTION**

106479 To submit a script is to create a batch job that executes the script. A script is submitted by a
 106480 request to a batch server. The *qsub* utility is a user-accessible batch client that submits a script.

106481 Upon successful completion, the *qsub* utility shall have created a batch job that will execute the
 106482 submitted script.

106483 The *qsub* utility shall submit a script by sending a *Queue Job Request* to a batch server.

106484 The *qsub* utility shall place the value of the following environment variables in the *Variable_List*
 106485 attribute of the batch job: *HOME*, *LANG*, *LOGNAME*, *PATH*, *MAIL*, *SHELL*, and *TZ*. The name of
 106486 the environment variable shall be the current name prefixed with the string *PBS_O_*.

106487 **Note:** If the current value of the *HOME* variable in the environment space of the *qsub* utility is
 106488 */aa/bb/cc*, then *qsub* shall place *PBS_O_HOME=/aa/bb/cc* in the *Variable_List* attribute of the
 106489 batch job.

106490 In addition to the variables described above, the *qsub* utility shall add the following variables
 106491 with the indicated values to the variable list:

106492 *PBS_O_WORKDIR* The absolute path of the current working directory of the *qsub* utility
 106493 process.

106494 *PBS_O_HOST* The name of the host on which the *qsub* utility is running.

106495 **OPTIONS**

106496 The *qsub* utility shall conform to XBD [Section 12.2](#) (on page 216).

106497 The following options shall be supported by the implementation:

106498 **-a *date_time*** Define the time at which a batch job becomes eligible for execution.

106499 The *qsub* utility shall accept an option-argument that conforms to the syntax of the
 106500 *time* operand of the *touch* utility.

106501

Table 4-19 Environment Variable Values (Utilities)

106502

106503

106504

106505

106506

106507

106508

106509

106510

106511

Variable Name	Value at qsub Time
<i>PBS_O_HOME</i>	<i>HOME</i>
<i>PBS_O_HOST</i>	Client host name
<i>PBS_O_LANG</i>	<i>LANG</i>
<i>PBS_O_LOGNAME</i>	<i>LOGNAME</i>
<i>PBS_O_PATH</i>	<i>PATH</i>
<i>PBS_O_MAIL</i>	<i>MAIL</i>
<i>PBS_O_SHELL</i>	<i>SHELL</i>
<i>PBS_O_TZ</i>	<i>TZ</i>
<i>PBS_O_WORKDIR</i>	Current working directory

106512

106513

Note: The server that initiates execution of the batch job will add other variables to the batch job's environment; see [Section 3.2.2.1](#) (on page 2433).

106514

106515

106516

106517

The *qsub* utility shall set the *Execution_Time* attribute of the batch job to the number of seconds since the Epoch that is equivalent to the local time expressed by the value of the *date_time* option-argument. The Epoch is defined in XBD [Section 3.150](#) (on page 57).

106518

106519

106520

If the *-a* option is not presented to the *qsub* utility, the utility shall set the *Execution_Time* attribute of the batch job to a time (number of seconds since the Epoch) that is earlier than the time at which the utility exits.

106521

-A account_string

106522

106523

Define the account to which the resource consumption of the batch job should be charged.

106524

The syntax of the *account_string* option-argument is unspecified.

106525

106526

The *qsub* utility shall set the *Account_Name* attribute of the batch job to the value of the *account_string* option-argument.

106527

106528

If the *-A* option is not presented to the *qsub* utility, the utility shall omit the *Account_Name* attribute from the attributes of the batch job.

106529

-c interval

Define whether the batch job should be checkpointed, and if so, how often.

106530

106531

The *qsub* utility shall accept a value for the interval option-argument that is one of the following:

106532

106533

n No checkpointing shall be performed on the batch job (NO_CHECKPOINT).

106534

106535

s Checkpointing shall be performed only when the batch server is shut down (CHECKPOINT_AT_SHUTDOWN).

106536

106537

106538

c Automatic periodic checkpointing shall be performed at the *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).

106539

106540

106541

106542

c=minutes Automatic periodic checkpointing shall be performed every *minutes* of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is greater. The *minutes* argument shall conform to the syntax for unsigned integers and shall be greater than zero.

106543

The *qsub* utility shall set the *Checkpoint* attribute of the batch job to the value of the

106544 *interval* option-argument.

106545 If the `-c` option is not presented to the *qsub* utility, the utility shall set the *Checkpoint*
 106546 attribute of the batch job to the single character 'u'
 106547 (CHECKPOINT_UNSPECIFIED).

106548 **-C** *directive_prefix*

106549 Define the prefix that declares a directive to the *qsub* utility within the script.

106550 The *directive_prefix* is not a batch job attribute; it affects the behavior of the *qsub*
 106551 utility.

106552 If the `-C` option is presented to the *qsub* utility, and the value of the *directive_prefix*
 106553 option-argument is the null string, the utility shall not scan the script file for
 106554 directives. If the `-C` option is not presented to the *qsub* utility, then the value of the
 106555 *PBS_DPREFIX* environment variable is used. If the environment variable is not
 106556 defined, then #PBS encoded in the portable character set is the default.

106557 **-e** *path_name*

106558 Define the path to be used for the standard error stream of the batch job.

106559 The *qsub* utility shall accept a *path_name* option-argument which can be preceded
 106560 by a host name element of the form *hostname*:

106561 If the *path_name* option-argument constitutes an absolute pathname, the *qsub* utility
 106562 shall set the *Error_Path* attribute of the batch job to the value of the *path_name*
 106563 option-argument.

106564 If the *path_name* option-argument constitutes a relative pathname and no host
 106565 name element is specified, the *qsub* utility shall set the *Error_Path* attribute of the
 106566 batch job to the value of the absolute pathname derived by expanding the
 106567 *path_name* option-argument relative to the current directory of the process
 106568 executing *qsub*.

106569 If the *path_name* option-argument constitutes a relative pathname and a host name
 106570 element is specified, the *qsub* utility shall set the *Error_Path* attribute of the batch
 106571 job to the value of the *path_name* option-argument without expansion. The host
 106572 name element shall be included.

106573 If the *path_name* option-argument does not include a host name element, the *qsub*
 106574 utility shall prefix the pathname with *hostname*:, where *hostname* is the name of the
 106575 host upon which the *qsub* utility is being executed.

106576 If the `-e` option is not presented to the *qsub* utility, the utility shall set the
 106577 *Error_Path* attribute of the batch job to the host name and path of the current
 106578 directory of the submitting process and the default filename.

106579 The default filename for standard error has the following format:

106580 *job_name.esquence_number*

106581 **-h** Specify that a USER hold is applied to the batch job.

106582 The *qsub* utility shall set the value of the *Hold_Types* attribute of the batch job to the
 106583 value USER.

106584 If the `-h` option is not presented to the *qsub* utility, the utility shall set the
 106585 *Hold_Types* attribute of the batch job to the value NO_HOLD.

106586 **-j** *join_list* Define which streams of the batch job are to be merged. The *qsub* **-j** option shall
 106587 accept a value for the *join_list* option-argument that is a string of alphanumeric
 106588 characters in the portable character set (see XBD [Section 6.1](#), on page 125).

106589 The *qsub* utility shall accept a *join_list* option-argument that consists of one or more
 106590 of the characters 'e' and 'o', or the single character 'n'.

106591 All of the other batch job output streams specified will be merged into the output
 106592 stream represented by the character listed first in the *join_list* option-argument.

106593 For each unique character in the *join_list* option-argument, the *qsub* utility shall
 106594 add a value to the *Join_Path* attribute of the batch job as follows, each representing
 106595 a different batch job stream to join:

- 106596 e The standard error of the batch job (JOIN_STD_ERROR).
- 106597 o The standard output of the batch job (JOIN_STD_OUTPUT).

106598 An existing *Join_Path* attribute can be cleared by the following join type:

106599 n NO_JOIN

106600 If 'n' is specified, then no files are joined. The *qsub* utility shall consider it an error
 106601 if any join type other than 'n' is combined with join type 'n'.

106602 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 106603 'n' within the *join_list* option-argument. The *qsub* utility shall permit the
 106604 repetition of characters, but shall not assign additional meaning to the repeated
 106605 characters.

106606 An implementation may define other join types. The conformance document for an
 106607 implementation shall describe any additional batch job streams, how they are
 106608 specified, their internal behavior, and how they affect the behavior of the utility.

106609 If the **-j** option is not presented to the *qsub* utility, the utility shall set the value of
 106610 the *Join_Path* attribute of the batch job to NO_JOIN.

106611 **-k** *keep_list* Define which output of the batch job to retain on the execution host.

106612 The *qsub* **-k** option shall accept a value for the *keep_list* option-argument that is a
 106613 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 106614 [6.1](#), on page 125).

106615 The *qsub* utility shall accept a *keep_list* option-argument that consists of one or
 106616 more of the characters 'e' and 'o', or the single character 'n'.

106617 For each unique character in the *keep_list* option-argument, the *qsub* utility shall
 106618 add a value to the *Keep_Files* attribute of the batch job as follows, each representing
 106619 a different batch job stream to keep:

- 106620 e The standard error of the batch job (KEEP_STD_ERROR).
- 106621 o The standard output of the batch job (KEEP_STD_OUTPUT).

106622 If both 'e' and 'o' are specified, then both files are retained. An existing
 106623 *Keep_Files* attribute can be cleared by the following keep type:

106624 n NO_KEEP

106625 If 'n' is specified, then no files are retained. The *qsub* utility shall consider it an
 106626 error if any keep type other than 'n' is combined with keep type 'n'.

106627 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 106628 'n' within the *keep_list* option-argument. The *qsub* utility shall permit the
 106629 repetition of characters, but shall not assign additional meaning to the repeated
 106630 characters.

106631 An implementation may define other keep types. The conformance document for
 106632 an implementation shall describe any additional keep types, how they are
 106633 specified, their internal behavior, and how they affect the behavior of the utility. If
 106634 the **-k** option is not presented to the *qsub* utility, the utility shall set the *Keep_Files*
 106635 attribute of the batch job to the value NO_KEEP.

106636 **-m** *mail_options*
 106637 Define the points in the execution of the batch job at which the batch server that
 106638 manages the batch job shall send mail about a change in the state of the batch job.

106639 The *qsub* **-m** option shall accept a value for the *mail_options* option-argument that
 106640 is a string of alphanumeric characters in the portable character set (see XBD [Section](#)
 106641 [6.1](#), on page 125).

106642 The *qsub* utility shall accept a value for the *mail_options* option-argument that is a
 106643 string of one or more of the characters 'e', 'b', and 'a', or the single character
 106644 'n'.

106645 For each unique character in the *mail_options* option-argument, the *qsub* utility shall
 106646 add a value to the *Mail_Users* attribute of the batch job as follows, each
 106647 representing a different time during the life of a batch job at which to send mail:

106648 e MAIL_AT_EXIT
 106649 b MAIL_AT_BEGINNING
 106650 a MAIL_AT_ABORT

106651 If any of these characters are duplicated in the *mail_options* option-argument, the
 106652 duplicates shall be ignored.

106653 An existing *Mail_Points* attribute can be cleared by the following mail type:

106654 n NO_MAIL

106655 If 'n' is specified, then mail is not sent. The *qsub* utility shall consider it an error if
 106656 any mail type other than 'n' is combined with mail type 'n'.

106657 Strictly conforming applications shall not repeat any of the characters 'e', 'b',
 106658 'a', or 'n' within the *mail_options* option-argument.

106659 The *qsub* utility shall permit the repetition of characters, but shall not assign
 106660 additional meaning to the repeated characters. An implementation may define
 106661 other mail types. The conformance document for an implementation shall describe
 106662 any additional mail types, how they are specified, their internal behavior, and how
 106663 they affect the behavior of the utility.

106664 If the **-m** option is not presented to the *qsub* utility, the utility shall set the
 106665 *Mail_Points* attribute to the value MAIL_AT_ABORT.

106666 **-M** *mail_list* Define the list of users to which a batch server that executes the batch job shall
 106667 send mail, if the server sends mail about the batch job.

106668 The syntax of the *mail_list* option-argument is unspecified.

106669 If the implementation of the *qsub* utility uses a name service to locate users, the

- 106670 utility should accept the syntax used by the name service.
- 106671 If the implementation of the *qsub* utility does not use a name service to locate users,
106672 the implementation should accept the following syntax for user names:
- 106673 *mail_address*[,*,mail_address*,, ...]
- 106674 The interpretation of *mail_address* is implementation-defined.
- 106675 The *qsub* utility shall set the *Mail_Users* attribute of the batch job to the value of the
106676 *mail_list* option-argument.
- 106677 If the **-M** option is not presented to the *qsub* utility, the utility shall place only the
106678 user name and host name for the current process in the *Mail_Users* attribute of the
106679 batch job.
- 106680 **-N name** Define the name of the batch job.
- 106681 The *qsub* **-N** option shall accept a value for the *name* option-argument that is a
106682 string of up to 15 alphanumeric characters in the portable character set (see XBD
106683 [Section 6.1](#), on page 125) where the first character is alphabetic.
- 106684 The *qsub* utility shall set the value of the *Job_Name* attribute of the batch job to the
106685 value of the *name* option-argument.
- 106686 If the **-N** option is not presented to the *qsub* utility, the utility shall set the *Job_Name*
106687 attribute of the batch job to the name of the *script* argument from which the
106688 directory specification if any, has been removed.
- 106689 If the **-N** option is not presented to the *qsub* utility, and the script is read from
106690 standard input, the utility shall set the *Job_Name* attribute of the batch job to the
106691 value STDIN.
- 106692 **-o path_name**
- 106693 Define the path for the standard output of the batch job.
- 106694 The *qsub* utility shall accept a *path_name* option-argument that conforms to the
106695 syntax of the *path_name* element defined in the System Interfaces volume of
106696 POSIX.1-2017, which can be preceded by a host name element of the form
106697 *hostname*..
- 106698 If the *path_name* option-argument constitutes an absolute pathname, the *qsub* utility
106699 shall set the *Output_Path* attribute of the batch job to the value of the *path_name*
106700 option-argument without expansion.
- 106701 If the *path_name* option-argument constitutes a relative pathname and no host
106702 name element is specified, the *qsub* utility shall set the *Output_Path* attribute of the
106703 batch job to the pathname derived by expanding the value of the *path_name* option-
106704 argument relative to the current directory of the process executing the *qsub*.
- 106705 If the *path_name* option-argument constitutes a relative pathname and a host name
106706 element is specified, the *qsub* utility shall set the *Output_Path* attribute of the batch
106707 job to the value of the *path_name* option-argument without expansion.
- 106708 If the *path_name* option-argument does not specify a host name element, the *qsub*
106709 utility shall prefix the pathname with *hostname*., where *hostname* is the name of the
106710 host upon which the *qsub* utility is executing.
- 106711 If the **-o** option is not presented to the *qsub* utility, the utility shall set the
106712 *Output_Path* attribute of the batch job to the host name and path of the current

106713 directory of the submitting process and the default filename.

106714 The default filename for standard output has the following format:

106715 *job_name.osequence_number*

106716 **-p priority** Define the priority the batch job should have relative to other batch jobs owned by
106717 the batch server.

106718 The *qsub* utility shall set the *Priority* attribute of the batch job to the value of the
106719 *priority* option-argument.

106720 If the **-p** option is not presented to the *qsub* utility, the value of the *Priority* attribute
106721 is implementation-defined.

106722 The *qsub* utility shall accept a value for the *priority* option-argument that conforms
106723 to the syntax for signed decimal integers, and which is not less than -1 024 and not
106724 greater than 1 023.

106725 **-q destination**

106726 Define the destination of the batch job.

106727 The destination is not a batch job attribute; it determines the batch server, and
106728 possibly the batch queue, to which the *qsub* utility batch queues the batch job.

106729 The *qsub* utility shall submit the script to the batch server named by the *destination*
106730 option-argument or the server that owns the batch queue named in the *destination*
106731 option-argument.

106732 The *qsub* utility shall accept an option-argument for the **-q** option that conforms to
106733 the syntax for a destination (see [Section 3.3.2](#), on page 2450).

106734 If the **-q** option is not presented to the *qsub* utility, the *qsub* utility shall submit the
106735 batch job to the default destination. The mechanism for determining the default
106736 destination is implementation-defined.

106737 **-r y | n** Define whether the batch job is rerunnable.

106738 If the value of the option-argument is *y*, the *qsub* utility shall set the *Rerunable*
106739 attribute of the batch job to TRUE.

106740 If the value of the option-argument is *n*, the *qsub* utility shall set the *Rerunable*
106741 attribute of the batch job to FALSE.

106742 If the **-r** option is not presented to the *qsub* utility, the utility shall set the *Rerunable*
106743 attribute of the batch job to TRUE.

106744 **-S path_name_list**

106745 Define the pathname to the shell under which the batch job is to execute.

106746 The *qsub* utility shall accept a *path_name_list* option-argument that conforms to the
106747 following syntax:

106748 *pathname[@host] [, , pathname[@host] , , . . .]*

106749 The *qsub* utility shall allow only one pathname for a given host name. The *qsub*
106750 utility shall allow only one pathname that is missing a corresponding host name.

106751 The *qsub* utility shall add a value to the *Shell_Path_List* attribute of the batch job for
106752 each entry in the *path_name_list* option-argument.

106753 If the **-S** option is not presented to the *qsub* utility, the utility shall set the

106754 *Shell_Path_List* attribute of the batch job to the null string.

106755 The conformance document for an implementation shall describe the mechanism
 106756 used to set the default shell and determine the current value of the default shell.
 106757 An implementation shall provide a means for the installation to set the default
 106758 shell to the login shell of the user under which the batch job is to execute. See
 106759 [Section 3.3.3](#) (on page 2451) for a means of removing *keyword=value* (and
 106760 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

106761 **-u** *user_list* Define the user name under which the batch job is to execute.

106762 The *qsub* utility shall accept a *user_list* option-argument that conforms to the
 106763 following syntax:

106764 `username[@host] [, , username[@host] , , . . .]`

106765 The *qsub* utility shall accept only one user name that is missing a corresponding
 106766 host name. The *qsub* utility shall accept only one user name per named host.

106767 The *qsub* utility shall add a value to the *User_List* attribute of the batch job for each
 106768 entry in the *user_list* option-argument.

106769 If the **-u** option is not presented to the *qsub* utility, the utility shall set the *User_List*
 106770 attribute of the batch job to the user name from which the utility is executing. See
 106771 [Section 3.3.3](#) (on page 2451) for a means of removing *keyword=value* (and
 106772 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

106773 **-v** *variable_list*

106774 Add to the list of variables that are exported to the session leader of the batch job.

106775 A *variable_list* is a set of strings of either the form *<variable>* or *<variable=value>*,
 106776 delimited by *<comma>* characters.

106777 If the **-v** option is presented to the *qsub* utility, the utility shall also add, to the
 106778 environment *Variable_List* attribute of the batch job, every variable named in the
 106779 environment *variable_list* option-argument and, optionally, values of specified
 106780 variables.

106781 If a value is not provided on the command line, the *qsub* utility shall set the value
 106782 of each variable in the environment *Variable_List* attribute of the batch job to the
 106783 value of the corresponding environment variable for the process in which the
 106784 utility is executing; see [Table 4-19](#) (on page 3179).

106785 A conforming application shall not repeat a variable in the environment
 106786 *variable_list* option-argument.

106787 The *qsub* utility shall not repeat a variable in the environment *Variable_List* attribute
 106788 of the batch job. See [Section 3.3.3](#) (on page 2451) for a means of removing
 106789 *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented
 106790 batch job attributes.

106791 **-V**

106792 Specify that all of the environment variables of the process are exported to the
 context of the batch job.

106793 The *qsub* utility shall place every environment variable in the process in which the
 106794 utility is executing in the list and shall set the value of each variable in the attribute
 106795 to the value of that variable in the process.

106796 **-z** Specify that the utility does not write the batch *job_identifier* of the created batch job
106797 to standard output.

106798 If the **-z** option is presented to the *qsub* utility, the utility shall not write the batch
106799 *job_identifier* of the created batch job to standard output.

106800 If the **-z** option is not presented to the *qsub* utility, the utility shall write the
106801 identifier of the created batch job to standard output.

106802 **OPERANDS**

106803 The *qsub* utility shall accept a *script* operand that indicates the path to the script of the batch job.

106804 If the *script* operand is not presented to the *qsub* utility, or if the operand is the single-character
106805 string '-', the utility shall read the script from standard input.

106806 If the script represents a partial path, the *qsub* utility shall expand the path relative to the current
106807 directory of the process executing the utility.

106808 **STDIN**

106809 The *qsub* utility reads the script of the batch job from standard input if the script operand is
106810 omitted or is the single character '-'.

106811 **INPUT FILES**

106812 In addition to binding the file indicated by the *script* operand to the batch job, the *qsub* utility
106813 reads the script file and acts on directives in the script.

106814 **ENVIRONMENT VARIABLES**

106815 The following environment variables shall affect the execution of *qsub*:

106816 **LANG** Provide a default value for the internationalization variables that are unset or null.
106817 (See XBD Section 8.2 (on page 174) the precedence of internationalization variables
106818 used to determine the values of locale categories.)

106819 **LC_ALL** If set to a non-empty string value, override the values of all the other
106820 internationalization variables.

106821 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
106822 characters (for example, single-byte as opposed to multi-byte characters in
106823 arguments).

106824 **LC_MESSAGES**

106825 Determine the locale that should be used to affect the format and contents of
106826 diagnostic messages written to standard error.

106827 **LOGNAME** Determine the login name of the user.

106828 **PBS_DPREFIX**

106829 Determine the default prefix for directives within the script.

106830 **SHELL** Determine the pathname of the preferred command language interpreter of the
106831 user.

106832 **TZ** Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
106833 unset or null, an unspecified default timezone shall be used.

106834 **ASYNCHRONOUS EVENTS**

106835 Once created, a batch job exists until it exits, aborts, or is deleted.

106836 After a batch job is created by the *qsub* utility, batch servers might route, execute, modify, or
106837 delete the batch job.

106838 **STDOUT**

106839 The *qsub* utility writes the batch *job_identifier* assigned to the batch job to standard output, unless
106840 the **-z** option is specified.

106841 **STDERR**

106842 The standard error shall be used only for diagnostic messages.

106843 **OUTPUT FILES**

106844 None.

106845 **EXTENDED DESCRIPTION**106846 **Script Preservation**

106847 The *qsub* utility shall make the script available to the server executing the batch job in such a
106848 way that the server executes the script as it exists at the time of submission.

106849 The *qsub* utility can send a copy of the script to the server with the *Queue Job Request* or store a
106850 temporary copy of the script in a location specified to the server.

106851 **Option Specification**

106852 A script can contain directives to the *qsub* utility.

106853 The *qsub* utility shall scan the lines of the script for directives, skipping blank lines, until the first
106854 line that begins with a string other than the directive string; if directives occur on subsequent
106855 lines, the utility shall ignore those directives.

106856 Lines are separated by a <newline>. If the first line of the script begins with "#!" or a <colon>
106857 (' : '), then it is skipped. The *qsub* utility shall process a line in the script as a directive if and
106858 only if the string of characters from the first non-white-space character on the line until the first
106859 <space> or <tab> on the line match the directive prefix. If a line in the script contains a directive
106860 and the final characters of the line are <backslash> and <newline>, then the next line shall be
106861 interpreted as a continuation of that directive.

106862 The *qsub* utility shall process the options and option-arguments contained on the directive prefix
106863 line using the same syntax as if the options were input on the *qsub* utility.

106864 The *qsub* utility shall continue to process a directive prefix line until after a <newline> is
106865 encountered. An implementation may ignore lines which, according to the syntax of the shell
106866 that will interpret the script, are comments. An implementation shall describe in the
106867 conformance document the format of any shell comments that it will recognize.

106868 If an option is present in both a directive and the arguments to the *qsub* utility, the utility shall
106869 ignore the option and the corresponding option-argument, if any, in the directive.

106870 If an option that is present in the directive is not present in the arguments to the *qsub* utility, the
106871 utility shall process the option and the option-argument, if any.

106872 In order of preference, the *qsub* utility shall select the directive prefix from one of the following
106873 sources:

106874 If the **-C** option is presented to the utility, the value of the *directive_prefix* option-argument

106875 If the environment variable *PBS_DPREFIX* is defined, the value of that variable

106876 The four-character string "#PBS" encoded in the portable character set

106877 If the **-C** option is present in the script file it shall be ignored.

106878 **EXIT STATUS**

106879 The following exit values shall be returned:

106880 0 Successful completion.

106881 >0 An error occurred.

106882 **CONSEQUENCES OF ERRORS**

106883 Default.

106884 **APPLICATION USAGE**

106885 None.

106886 **EXAMPLES**

106887 None.

106888 **RATIONALE**

106889 The *qsub* utility allows users to create a batch job that will process the script specified as the
106890 operand of the utility.

106891 The options of the *qsub* utility allow users to control many aspects of the queuing and execution
106892 of a batch job.

106893 The **-a** option allows users to designate the time after which the batch job will become eligible to
106894 run. By specifying an execution time, users can take advantage of resources at off-peak hours,
106895 synchronize jobs with chronologically predictable events, and perhaps take advantage of off-
106896 peak pricing of computing time. For these reasons and others, a timing option is existing
106897 practice on the part of almost every batch system, including NQS.

106898 The **-A** option allows users to specify the account that will be charged for the batch job. Support
106899 for account is not mandatory for conforming batch servers.

106900 The **-C** option allows users to prescribe the prefix for directives within the script file. The default
106901 prefix "#PBS" may be inappropriate if the script will be interpreted with an alternate shell, as
106902 specified by the **-S** option.

106903 The **-c** option allows users to establish the checkpointing interval for their jobs. A checkpointing
106904 system, which is not defined by this volume of POSIX.1-2017, allows recovery of a batch job at
106905 the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that
106906 consume expensive computing time or must meet a critical schedule. Users should be allowed to
106907 make the tradeoff between the overhead of checkpointing and the risk to the timely completion
106908 of the batch job; therefore, this volume of POSIX.1-2017 provides the checkpointing interval
106909 option. Support for checkpointing is optional for batch servers.

106910 The **-e** option allows users to redirect the standard error streams of their jobs to a non-default
106911 path. For example, if the submitted script generally produces a great deal of useless error
106912 output, a user might redirect the standard error output to the null device. Or, if the file system
106913 holding the default location (the home directory of the user) has too little free space, the user
106914 might redirect the standard error stream to a file in another file system.

106915 The **-h** option allows users to create a batch job that is held until explicitly released. The ability
106916 to create a held job is useful when some external event must complete before the batch job can
106917 execute. For example, the user might submit a held job and release it when the system load has
106918 dropped.

106919 The **-j** option allows users to merge the standard error of a batch job into its standard output
106920 stream, which has the advantage of showing the sequential relationship between output and
106921 error messages.

106922 The **-m** option allows users to designate those points in the execution of a batch job at which
106923 mail will be sent to the submitting user, or to the account(s) indicated by the **-M** option. By
106924 requesting mail notification at points of interest in the life of a job, the submitting user, or other
106925 designated users, can track the progress of a batch job.

106926 The **-N** option allows users to associate a name with the batch job. The job name in no way
106927 affects the processing of the batch job, but rather serves as a mnemonic handle for users. For
106928 example, the batch job name can help the user distinguish between multiple jobs listed by the
106929 *qstat* utility.

106930 The **-o** option allows users to redirect the standard output stream. A user might, for example,
106931 wish to redirect to the null device the standard output stream of a job that produces copious yet
106932 superfluous output.

106933 The **-P** option allows users to designate the relative priority of a batch job for selection from a
106934 queue.

106935 The **-q** option allows users to specify an initial queue for the batch job. If the user specifies a
106936 routing queue, the batch server routes the batch job to another queue for execution or further
106937 routing. If the user specifies a non-routing queue, the batch server of the queue eventually
106938 executes the batch job.

106939 The **-r** option allows users to control whether the submitted job will be rerun if the controlling
106940 batch node fails during execution of the batch job. The **-r** option likewise allows users to
106941 indicate whether or not the batch job is eligible to be rerun by the *qrerun* utility. Some jobs cannot
106942 be correctly rerun because of changes they make in the state of databases or other aspects of
106943 their environment. This volume of POSIX.1-2017 specifies that the default, if the **-r** option is not
106944 presented to the utility, will be that the batch job cannot be rerun, since the result of rerunning a
106945 non-rerunnable job might be catastrophic.

106946 The **-S** option allows users to specify the program (usually a shell) that will be invoked to
106947 process the script of the batch job. This option has been modified to allow a list of shell names
106948 and locations associated with different hosts.

106949 The **-u** option is useful when the submitting user is authorized to use more than one account on
106950 a given host, in which case the **-u** option allows the user to select from among those accounts.
106951 The option-argument is a list of user-host pairs, so that the submitting user can provide different
106952 user identifiers for different nodes in the event the batch job is routed. The **-u** option provides a
106953 lot of flexibility to accommodate sites with complex account structures. Users that have the same
106954 user identifier on all the hosts they are authorized to use will not need to use the **-u** option.

106955 The **-V** option allows users to export all their current environment variables, as of the time the
106956 batch job is submitted, to the context of the processes of the batch job.

106957 The **-v** option allows users to export specific environment variables from their current process to
106958 the processes of the batch job.

106959 The **-z** option allows users to suppress the writing of the batch job identifier to standard output.
106960 The **-z** option is an existing NQS practice that has been standardized.

106961 Historically, the *qsub* utility has served the batch job-submission function in the NQS system, the
106962 existing practice on which it is based. Some changes and additions have been made to the *qsub*
106963 utility in this volume of POSIX.1-2017, *vis-a-vis* NQS, as a result of the growing pool of
106964 experience with distributed batch systems.

106965 The set of features of the *qsub* utility as defined in this volume of POSIX.1-2017 appears to
106966 incorporate all the common existing practice on potentially conforming platforms.

106967 FUTURE DIRECTIONS

106968 The *qsub* utility may be removed in a future version.

106969 SEE ALSO

106970 [Chapter 3](#) (on page 2427), [*qrerun*](#), [*qstat*](#), [*touch*](#)

106971 XBD [Section 3.150](#) (on page 57), [Section 6.1](#) (on page 125), [Chapter 8](#) (on page 173), [Section 12.2](#)
106972 (on page 216)

106973 CHANGE HISTORY

106974 Derived from IEEE Std 1003.2d-1994.

106975 Issue 6

106976 The `-I` option has been removed as there is no portable description of the resources that are
106977 allowed or required by the batch job.

106978 Issue 7

106979 The *qsub* utility is marked obsolescent.

106980 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

106981 **NAME**

106982 read — read from standard input into shell variables

106983 **SYNOPSIS**

106984 read [-r] var...

106985 **DESCRIPTION**106986 The *read* utility shall read a single logical line from standard input into one or more shell
106987 variables.106988 By default, unless the *-r* option is specified, `<backslash>` shall act as an escape character. An
106989 unescaped `<backslash>` shall preserve the literal value of the following character, with the
106990 exception of a `<newline>`. If a `<newline>` follows the `<backslash>`, the *read* utility shall interpret
106991 this as line continuation. The `<backslash>` and `<newline>` shall be removed before splitting the
106992 input into fields. All other unescaped `<backslash>` characters shall be removed after splitting the
106993 input into fields.106994 If standard input is a terminal device and the invoking shell is interactive, *read* shall prompt for a
106995 continuation line when it reads an input line ending with a `<backslash>` `<newline>`, unless the
106996 *-r* option is specified.106997 The terminating `<newline>` (if any) shall be removed from the input and the results shall be split
106998 into fields as in the shell for the results of parameter expansion (see [Section 2.6.5](#), on page 2359);
106999 the first field shall be assigned to the first variable *var*, the second field to the second variable
107000 *var*, and so on. If there are fewer fields than there are *var* operands, the remaining *vars* shall be
107001 set to empty strings. If there are fewer *vars* than fields, the last *var* shall be set to a value
107002 comprising the following elements:107003 The field that corresponds to the last *var* in the normal assignment sequence described
107004 above107005 The delimiter(s) that follow the field corresponding to the last *var*107006 The remaining fields and their delimiters, with trailing *IFS* white space ignored107007 The setting of variables specified by the *var* operands shall affect the current shell execution
107008 environment; see [Section 2.12](#) (on page 2381). If it is called in a subshell or separate utility
107009 execution environment, such as one of the following:107010 (read foo)
107011 nohup read ...
107012 find . -exec read ... \;

107013 it shall not affect the shell variables in the caller's environment.

107014 **OPTIONS**107015 The *read* utility shall conform to XBD [Section 12.2](#) (on page 216).

107016 The following option is supported:

107017 *-r* Do not treat a `<backslash>` character in any special way. Consider each
107018 `<backslash>` to be part of the input line.107019 **OPERANDS**

107020 The following operand shall be supported:

107021 *var* The name of an existing or nonexisting shell variable.

107022 **STDIN**

107023 The standard input shall be a text file.

107024 **INPUT FILES**

107025 None.

107026 **ENVIRONMENT VARIABLES**107027 The following environment variables shall affect the execution of *read*:107028 *IFS* Determine the internal field separators used to delimit fields; see [Section 2.5.3](#) (on
107029 page 2351).107030 *LANG* Provide a default value for the internationalization variables that are unset or null.
107031 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
107032 variables used to determine the values of locale categories.)107033 *LC_ALL* If set to a non-empty string value, override the values of all the other
107034 internationalization variables.107035 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
107036 characters (for example, single-byte as opposed to multi-byte characters in
107037 arguments).107038 *LC_MESSAGES*107039 Determine the locale that should be used to affect the format and contents of
107040 diagnostic messages written to standard error.107041 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.107042 *PS2* Provide the prompt string that an interactive shell shall write to standard error
107043 when a line ending with a <backslash> <newline> is read and the *-r* option was
107044 not specified.107045 **ASYNCHRONOUS EVENTS**

107046 Default.

107047 **STDOUT**

107048 Not used.

107049 **STDERR**

107050 The standard error shall be used for diagnostic messages and prompts for continued input.

107051 **OUTPUT FILES**

107052 None.

107053 **EXTENDED DESCRIPTION**

107054 None.

107055 **EXIT STATUS**

107056 The following exit values shall be returned:

107057 0 Successful completion.

107058 >0 End-of-file was detected or an error occurred.

107059 **CONSEQUENCES OF ERRORS**

107060 Default.

107061 APPLICATION USAGE

107062 The `-r` option is included to enable *read* to subsume the purpose of the *line* utility, which is not
107063 included in POSIX.1-2017.

107064 EXAMPLES

107065 The following command:

```
107066 while read -r xx yy  
107067 do  
107068     printf "%s %s\n" "$yy" "$xx"  
107069 done < input_file
```

107070 prints a file with the first field of each line moved to the end of the line.

107071 RATIONALE

107072 The *read* utility historically has been a shell built-in. It was separated off into its own utility to
107073 take advantage of the richer description of functionality introduced by this volume of
107074 POSIX.1-2017.

107075 Since *read* affects the current shell execution environment, it is generally provided as a shell
107076 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
107077 of the following:

```
107078 (read foo)  
107079 nohup read ...  
107080 find . -exec read ... \;
```

107081 it does not affect the shell variables in the environment of the caller.

107082 Although the standard input is required to be a text file, and therefore will always end with a
107083 <newline> (unless it is an empty file), the processing of continuation lines when the `-r` option is
107084 not used can result in the input not ending with a <newline>. This occurs if the last line of the
107085 input file ends with a <backslash> <newline>. It is for this reason that “if any” is used in “The
107086 terminating <newline> (if any) shall be removed from the input” in the description. It is not a
107087 relaxation of the requirement for standard input to be a text file.

107088 FUTURE DIRECTIONS

107089 None.

107090 SEE ALSO

107091 [Chapter 2](#) (on page 2345)

107092 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

107093 CHANGE HISTORY

107094 First released in Issue 2.

107095 Issue 7

107096 Austin Group Interpretation 1003.1-2001 #194 is applied, clarifying the handling of the
107097 <backslash> escape character.

107098 SD5-XCU-ERN-126 is applied, clarifying that input lines end with a <newline>.

107099 The description of here-documents is removed from the *read* reference page.

107100 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0162 [958] is applied.

107101 **NAME**

107102 renice — set nice values of running processes

107103 **SYNOPSIS**107104 renice [-g|-p|-u] -n *increment* *ID...*107105 **DESCRIPTION**

107106 The *renice* utility shall request that the nice values (see XBD [Section 3.244](#), on page 72) of one or
 107107 more running processes be changed. By default, the applicable processes are specified by their
 107108 process IDs. When a process group is specified (see **-g**), the request shall apply to all processes
 107109 in the process group.

107110 The nice value shall be bounded in an implementation-defined manner. If the requested
 107111 *increment* would raise or lower the nice value of the executed utility beyond implementation-
 107112 defined limits, then the limit whose value was exceeded shall be used.

107113 When a user is *reniced*, the request applies to all processes whose saved set-user-ID matches the
 107114 user ID corresponding to the user.

107115 Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values
 107116 of any process unless the user requesting such a change has appropriate privileges to do so for
 107117 the specified process. If the user lacks appropriate privileges to perform the requested action, the
 107118 utility shall return an error status.

107119 The saved set-user-ID of the user's process shall be checked instead of its effective user ID when
 107120 *renice* attempts to determine the user ID of the process in order to determine whether the user
 107121 has appropriate privileges.

107122 **OPTIONS**107123 The *renice* utility shall conform to XBD [Section 12.2](#) (on page 216), except for Guideline 9.

107124 The following options shall be supported:

107125 **-g** Interpret the following operands as unsigned decimal integer process group IDs.

107126 **-n** *increment* Specify how the nice value of the specified process or processes is to be adjusted.
 107127 The *increment* option-argument is a positive or negative decimal integer that shall
 107128 be used to modify the nice value of the specified process or processes.

107129 Positive *increment* values shall cause a lower nice value. Negative *increment* values
 107130 may require appropriate privileges and shall cause a higher nice value.

107131 **-p** Interpret the following operands as unsigned decimal integer process IDs. The **-p**
 107132 option is the default if no options are specified.

107133 **-u** Interpret the following operands as users. If a user exists with a user name equal to
 107134 the operand, then the user ID of that user is used in further processing. Otherwise,
 107135 if the operand represents an unsigned decimal integer, it shall be used as the
 107136 numeric user ID of the user.

107137 **OPERANDS**

107138 The following operands shall be supported:

107139 *ID* A process ID, process group ID, or user name/user ID, depending on the option
 107140 selected.

107141 **STDIN**

107142 Not used.

107143 **INPUT FILES**

107144 None.

107145 **ENVIRONMENT VARIABLES**107146 The following environment variables shall affect the execution of *renice*:

107147 *LANG* Provide a default value for the internationalization variables that are unset or null.
 107148 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 107149 variables used to determine the values of locale categories.)

107150 *LC_ALL* If set to a non-empty string value, override the values of all the other
 107151 internationalization variables.

107152 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 107153 characters (for example, single-byte as opposed to multi-byte characters in
 107154 arguments).

107155 *LC_MESSAGES*

107156 Determine the locale that should be used to affect the format and contents of
 107157 diagnostic messages written to standard error.

107158 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

107159 **ASYNCHRONOUS EVENTS**

107160 Default.

107161 **STDOUT**

107162 Not used.

107163 **STDERR**

107164 The standard error shall be used only for diagnostic messages.

107165 **OUTPUT FILES**

107166 None.

107167 **EXTENDED DESCRIPTION**

107168 None.

107169 **EXIT STATUS**

107170 The following exit values shall be returned:

107171 0 Successful completion.

107172 >0 An error occurred.

107173 **CONSEQUENCES OF ERRORS**

107174 Default.

107175 **APPLICATION USAGE**

107176 None.

107177 **EXAMPLES**

107178 1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

107179 `renice -n 5 -p 987 32`

107180 2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the
 107181 user has appropriate privileges to do so:

107182 `renice -n -4 -g 324 76`

107183 3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice
107184 value:

```
107185 renice -n 4 -u 8 sas
```

107186 Useful nice value increments on historical systems include 19 or 20 (the affected processes run
107187 only when nothing else in the system attempts to run) and any negative number (to make
107188 processes run faster).

107189 RATIONALE

107190 The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-
107191 argument. However, for clarity, they have been included in the OPTIONS section, rather than
107192 the OPERANDS section.

107193 The definition of nice value is not intended to suggest that all processes in a system have
107194 priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the
107195 System Interfaces volume of POSIX.1-2017 make the notion of a single underlying priority for all
107196 scheduling policies problematic. Some implementations may implement the *nice*-related features
107197 to affect all processes on the system, others to affect just the general time-sharing activities
107198 implied by this volume of POSIX.1-2017, and others may have no effect at all. Because of the use
107199 of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are
107200 possible.

107201 Originally, this utility was written in the historical manner, using the term “nice value”. This
107202 was always a point of concern with users because it was never intuitively obvious what this
107203 meant. With a newer version of *renice*, which used the term “system scheduling priority”, it was
107204 hoped that novice users could better understand what this utility was meant to do. Also, it
107205 would be easier to document what the utility was meant to do. Unfortunately, the addition of
107206 the POSIX realtime scheduling capabilities introduced the concepts of process and thread
107207 scheduling priorities that were totally unaffected by the *nice/renice* utilities or the
107208 *nice()/setpriority()* functions. Continuing to use the term “system scheduling priority” would
107209 have incorrectly suggested that these utilities and functions were indeed affecting these realtime
107210 priorities. It was decided to revert to the historical term “nice value” to reference this unrelated
107211 process attribute.

107212 Although this utility has use by system administrators (and in fact appears in the system
107213 administration portion of the BSD documentation), the standard developers considered that it
107214 was very useful for individual end users to control their own processes.

107215 Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
107216 renice nice_value[-p] pid...[-g gid...][-p pid...][-u user...]  
107217 renice nice_value -g gid...[-g gid...]-p pid...[-u user...]  
107218 renice nice_value -u user...[-g gid...]-p pid...[-u user...]
```

107219 These forms are no longer specified by POSIX.1-2017 but may be present in some
107220 implementations.

107221 FUTURE DIRECTIONS

107222 None.

107223 SEE ALSO

107224 [nice](#)

107225 XBD [Section 3.244](#) (on page 72), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

107226 **CHANGE HISTORY**

107227 First released in Issue 4.

107228 **Issue 5**

107229 In the SYNOPSIS, an ellipsis is added to the `-u` option in all three obsolescent forms.

107230 **Issue 6**

107231 This utility is marked as part of the User Portability Utilities option.

107232 The APPLICATION USAGE section is added.

107233 The obsolescent forms of the SYNOPSIS are removed.

107234 Text previously conditional on POSIX_SAVED_IDS is mandatory in this version. This is a FIPS
107235 requirement.

107236 **Issue 7**

107237 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility
107238 Syntax Guidelines does not apply.

107239 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

107240 The *renice* utility is moved from the User Portability Utilities option to the Base. User Portability
107241 Utilities is now an option for interactive utilities.

107242 **NAME**

107243 rm — remove directory entries

107244 **SYNOPSIS**107245 rm [-iRr] *file...*107246 rm -f [-iRr] [*file...*]107247 **DESCRIPTION**107248 The *rm* utility shall remove the directory entry specified by each *file* argument.

107249 If either of the files dot or dot-dot are specified as the basename portion of an operand (that is,
 107250 the final pathname component) or if an operand resolves to the root directory, *rm* shall write a
 107251 diagnostic message to standard error and do nothing more with such operands.

107252 For each *file* the following steps shall be taken:

- 107253 1. If the *file* does not exist:
 - 107254 a. If the `-f` option is not specified, *rm* shall write a diagnostic message to standard
 107255 error.
 - 107256 b. Go on to any remaining *files*.
 - 107257 2. If *file* is of type directory, the following steps shall be taken:
 - 107258 a. If neither the `-R` option nor the `-r` option is specified, *rm* shall write a diagnostic
 107259 message to standard error, do nothing more with *file*, and go on to any remaining
 107260 files.
 - 107261 b. If *file* is an empty directory, *rm* may skip to step 2d. If the `-f` option is not specified,
 107262 and either the permissions of *file* do not permit writing and the standard input is a
 107263 terminal or the `-i` option is specified, *rm* shall write a prompt to standard error and
 107264 read a line from the standard input. If the response is not affirmative, *rm* shall do
 107265 nothing more with the current file and go on to any remaining files.
 - 107266 c. For each entry contained in *file*, other than dot or dot-dot, the four steps listed here
 107267 (1 to 4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall
 107268 not traverse directories by following symbolic links into other parts of the
 107269 hierarchy, but shall remove the links themselves.
 - 107270 d. If the `-i` option is specified, *rm* shall write a prompt to standard error and read a
 107271 line from the standard input. If the response is not affirmative, *rm* shall do nothing
 107272 more with the current file, and go on to any remaining files.
 - 107273 3. If *file* is not of type directory, the `-f` option is not specified, and either the permissions of
 107274 *file* do not permit writing and the standard input is a terminal or the `-i` option is specified,
 107275 *rm* shall write a prompt to the standard error and read a line from the standard input. If
 107276 the response is not affirmative, *rm* shall do nothing more with the current file and go on
 107277 to any remaining files.
 - 107278 4. If the current file is a directory, *rm* shall perform actions equivalent to the *rmdir()*
 107279 function defined in the System Interfaces volume of POSIX.1-2017 called with a pathname of the
 107280 current file used as the *path* argument. If the current file is not a directory, *rm* shall
 107281 perform actions equivalent to the *unlink()* function defined in the System Interfaces
 107282 volume of POSIX.1-2017 called with a pathname of the current file used as the *path*
 107283 argument.
- 107284 If this fails for any reason, *rm* shall write a diagnostic message to standard error, do
 107285 nothing more with the current file, and go on to any remaining files.

107286 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail
 107287 due to path length limitations (unless an operand specified by the user exceeds system
 107288 limitations).

107289 OPTIONS

107290 The *rm* utility shall conform to XBD [Section 12.2](#) (on page 216).

107291 The following options shall be supported:

107292 **-f** Do not prompt for confirmation. Do not write diagnostic messages or modify the
 107293 exit status in the case of no file operands, or in the case of operands that do not
 107294 exist. Any previous occurrences of the **-i** option shall be ignored.

107295 **-i** Prompt for confirmation as described previously. Any previous occurrences of the
 107296 **-f** option shall be ignored.

107297 **-R** Remove file hierarchies. See the DESCRIPTION.

107298 **-r** Equivalent to **-R**.

107299 OPERANDS

107300 The following operand shall be supported:

107301 *file* A pathname of a directory entry to be removed.

107302 STDIN

107303 The standard input shall be used to read an input line in response to each prompt specified in
 107304 the STDOUT section. Otherwise, the standard input shall not be used.

107305 INPUT FILES

107306 None.

107307 ENVIRONMENT VARIABLES

107308 The following environment variables shall affect the execution of *rm*:

107309 *LANG* Provide a default value for the internationalization variables that are unset or null.
 107310 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 107311 variables used to determine the values of locale categories.)

107312 *LC_ALL* If set to a non-empty string value, override the values of all the other
 107313 internationalization variables.

107314 *LC_COLLATE*

107315 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 107316 character collating elements used in the extended regular expression defined for
 107317 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

107318 *LC_CTYPE*

107319 Determine the locale for the interpretation of sequences of bytes of text data as
 107320 characters (for example, single-byte as opposed to multi-byte characters in
 107321 arguments) and the behavior of character classes within regular expressions used
 107322 in the extended regular expression defined for the **yesexpr** locale keyword in the
LC_MESSAGES category.

107323 *LC_MESSAGES*

107324 Determine the locale used to process affirmative responses, and the locale used to
 107325 affect the format and contents of diagnostic messages and prompts written to
 107326 standard error.

107327 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

107328 **ASYNCHRONOUS EVENTS**

107329 Default.

107330 **STDOUT**

107331 Not used.

107332 **STDERR**

107333 Prompts shall be written to standard error under the conditions specified in the **DESCRIPTION**

107334 and **OPTIONS** sections. The prompts shall contain the *file* pathname, but their format is

107335 otherwise unspecified. The standard error also shall be used for diagnostic messages.

107336 **OUTPUT FILES**

107337 None.

107338 **EXTENDED DESCRIPTION**

107339 None.

107340 **EXIT STATUS**

107341 The following exit values shall be returned:

107342 0 Each directory entry was successfully removed, unless its removal was canceled by a non-

107343 affirmative response to a prompt for confirmation.

107344 >0 An error occurred.

107345 **CONSEQUENCES OF ERRORS**

107346 Default.

107347 **APPLICATION USAGE**

107348 The *rm* utility is forbidden to remove the names *dot* and *dot-dot* in order to avoid the

107349 consequences of inadvertently doing something like:

107350 `rm -r .*`

107351 Some implementations do not permit the removal of the last link to an executable binary file that

107352 is being executed; see the [EBUSY] error in the *unlink()* function defined in the System Interfaces

107353 volume of POSIX.1-2017. Thus, the *rm* utility can fail to remove such files.

107354 The *-i* option causes *rm* to prompt and read the standard input even if the standard input is not

107355 a terminal, but in the absence of *-i* the mode prompting is not done when the standard input is

107356 not a terminal.

107357 **EXAMPLES**

107358 1. The following command:

107359 `rm a.out core`

107360 removes the directory entries: **a.out** and **core**.

107361 2. The following command:

107362 `rm -Rf junk`

107363 removes the directory **junk** and all its contents, without prompting.

107364 **RATIONALE**

107365 For absolute clarity, paragraphs (2b) and (3) in the **DESCRIPTION** of *rm* describing the behavior

107366 when prompting for confirmation, should be interpreted in the following manner:

107367 `if ((NOT f_option) AND`

107368 ((not_writable AND input_is_terminal) OR i_option))

107369 The exact format of the interactive prompts is unspecified. Only the general nature of the
 107370 contents of prompts are specified because implementations may desire more descriptive
 107371 prompts than those used on historical implementations. Therefore, an application not using the
 107372 `-f` option, or using the `-i` option, relies on the system to provide the most suitable dialog directly
 107373 with the user, based on the behavior specified.

107374 The `-r` option is historical practice on all known systems. The synonym `-R` option is provided
 107375 for consistency with the other utilities in this volume of POSIX.1-2017 that provide options
 107376 requesting recursive descent through the file hierarchy.

107377 The behavior of the `-f` option in historical versions of *rm* is inconsistent. In general, along with
 107378 “forcing” the unlink without prompting for permission, it always causes diagnostic messages to
 107379 be suppressed and the exit status to be unmodified for nonexistent operands and files that
 107380 cannot be unlinked. In some versions, however, the `-f` option suppresses usage messages and
 107381 system errors as well. Suppressing such messages is not a service to either shell scripts or users.

107382 It is less clear that error messages regarding files that cannot be unlinked (removed) should be
 107383 suppressed. Although this is historical practice, this volume of POSIX.1-2017 does not permit the
 107384 `-f` option to suppress such messages.

107385 When given the `-r` and `-i` options, historical versions of *rm* prompt the user twice for each
 107386 directory, once before removing its contents and once before actually attempting to delete the
 107387 directory entry that names it. This allows the user to “prune” the file hierarchy walk. Historical
 107388 versions of *rm* were inconsistent in that some did not do the former prompt for directories
 107389 named on the command line and others had obscure prompting behavior when the `-i` option
 107390 was specified and the permissions of the file did not permit writing. The POSIX Shell and
 107391 Utilities *rm* differs little from historic practice, but does require that prompts be consistent.
 107392 Historical versions of *rm* were also inconsistent in that prompts were done to both standard
 107393 output and standard error. This volume of POSIX.1-2017 requires that prompts be done to
 107394 standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that
 107395 provide an option to list deleted files on standard output.

107396 The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be
 107397 deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its
 107398 descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the
 107399 historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted
 107400 to fail because of path length restrictions, unless an operand specified by the user is longer than
 107401 {PATH_MAX}.

107402 The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of
 107403 the dependence on the *unlink()* functionality, per the DESCRIPTION. When removing
 107404 hierarchies with `-r` or `-R`, the prohibition on following symbolic links has to be made explicit.

107405 **FUTURE DIRECTIONS**

107406 None.

107407 **SEE ALSO**

107408 *rmdir*

107409 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

107410 XSH *remove()*, *rmdir()*, *unlink()*

107411 **CHANGE HISTORY**

107412 First released in Issue 2.

107413 **Issue 5**

107414 The FUTURE DIRECTIONS section is added.

107415 **Issue 6**

107416 Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft
107417 standard.

107418 **Issue 7**

107419 Austin Group Interpretations 1003.1-2001 #019 and #091 are applied.

107420 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
107421 *LC_MESSAGES* environment variable.

107422 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0163 [542], XCU/TC2-2008/0164
107423 [819], and XCU/TC2-2008/0165 [542] are applied.

107424 **NAME**107425 rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)107426 **SYNOPSIS**107427 XSI `rmdel -r SID file...`107428 **DESCRIPTION**

107429 The *rmdel* utility shall remove the delta specified by the *SID* from each named SCCS file. The
 107430 delta to be removed shall be the most recent delta in its branch in the delta chain of each named
 107431 SCCS file. In addition, the application shall ensure that the *SID* specified is not that of a version
 107432 being edited for the purpose of making a delta; that is, if a *p-file* (see *get*) exists for the named
 107433 SCCS file, the *SID* specified shall not appear in any entry of the *p-file*.

107434 Removal of a delta shall be restricted to:

- 107435 1. The user who made the delta
- 107436 2. The owner of the SCCS file
- 107437 3. The owner of the directory containing the SCCS file

107438 **OPTIONS**107439 The *rmdel* utility shall conform to XBD [Section 12.2](#) (on page 216).

107440 The following option shall be supported:

107441 **-r** *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.107442 **OPERANDS**

107443 The following operand shall be supported:

107444 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rmdel*
 107445 utility shall behave as though each file in the directory were specified as a named
 107446 file, except that non-SCCS files (last component of the pathname does not begin
 107447 with **s**.) and unreadable files shall be silently ignored.

107448 If exactly one *file* operand appears, and it is '-', the standard input shall be read;
 107449 each line of the standard input is taken to be the name of an SCCS file to be
 107450 processed. Non-SCCS files and unreadable files shall be silently ignored.

107451 **STDIN**

107452 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each
 107453 line of the text file shall be interpreted as an SCCS pathname.

107454 **INPUT FILES**

107455 The SCCS files shall be files of unspecified format.

107456 **ENVIRONMENT VARIABLES**107457 The following environment variables shall affect the execution of *rmdel*:

107458 **LANG** Provide a default value for the internationalization variables that are unset or null.
 107459 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 107460 variables used to determine the values of locale categories.)

107461 **LC_ALL** If set to a non-empty string value, override the values of all the other
 107462 internationalization variables.

107463 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 107464 characters (for example, single-byte as opposed to multi-byte characters in
 107465 arguments and input files).

- 107466 *LC_MESSAGES*
107467 Determine the locale that should be used to affect the format and contents of
107468 diagnostic messages written to standard error.
- 107469 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 107470 **ASYNCHRONOUS EVENTS**
107471 Default.
- 107472 **STDOUT**
107473 Not used.
- 107474 **STDERR**
107475 The standard error shall be used only for diagnostic messages.
- 107476 **OUTPUT FILES**
107477 The SCCS files shall be files of unspecified format. During processing of a *file*, a temporary *x-file*,
107478 as described in *admin*, may be created and deleted; a locking *z-file*, as described in *get*, may be
107479 created and deleted.
- 107480 **EXTENDED DESCRIPTION**
107481 None.
- 107482 **EXIT STATUS**
107483 The following exit values shall be returned:
107484 0 Successful completion.
107485 >0 An error occurred.
- 107486 **CONSEQUENCES OF ERRORS**
107487 Default.
- 107488 **APPLICATION USAGE**
107489 None.
- 107490 **EXAMPLES**
107491 None.
- 107492 **RATIONALE**
107493 None.
- 107494 **FUTURE DIRECTIONS**
107495 None.
- 107496 **SEE ALSO**
107497 *admin, delta, get, prs*
107498 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 107499 **CHANGE HISTORY**
107500 First released in Issue 2.
- 107501 **Issue 6**
107502 The normative text is reworded to avoid use of the term “must” for application requirements.

107503 **NAME**107504 `rmdir` — remove directories107505 **SYNOPSIS**107506 `rmdir [-p] dir...`107507 **DESCRIPTION**107508 The *rmdir* utility shall remove the directory entry specified by each *dir* operand.107509 For each *dir* operand, the *rmdir* utility shall perform actions equivalent to the *rmdir()* function
107510 called with the *dir* operand as its only argument.107511 Directories shall be processed in the order specified. If a directory and a subdirectory of that
107512 directory are specified in a single invocation of the *rmdir* utility, the application shall specify the
107513 subdirectory before the parent directory so that the parent directory will be empty when the
107514 *rmdir* utility tries to remove it.107515 **OPTIONS**107516 The *rmdir* utility shall conform to XBD [Section 12.2](#) (on page 216).

107517 The following option shall be supported:

107518 **-p** Remove all directories in a pathname. For each *dir* operand:

- 107519 1. The directory entry it names shall be removed.
- 107520 2. If the *dir* operand includes more than one pathname component, effects
107521 equivalent to the following command shall occur:

107522 `rmdir -p $(dirname dir)`107523 **OPERANDS**

107524 The following operand shall be supported:

107525 *dir* A pathname of an empty directory to be removed.107526 **STDIN**

107527 Not used.

107528 **INPUT FILES**

107529 None.

107530 **ENVIRONMENT VARIABLES**107531 The following environment variables shall affect the execution of *rmdir*:107532 *LANG* Provide a default value for the internationalization variables that are unset or null.
107533 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
107534 variables used to determine the values of locale categories.)107535 *LC_ALL* If set to a non-empty string value, override the values of all the other
107536 internationalization variables.107537 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
107538 characters (for example, single-byte as opposed to multi-byte characters in
107539 arguments).107540 *LC_MESSAGES*107541 Determine the locale that should be used to affect the format and contents of
107542 diagnostic messages written to standard error.

- 107543 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 107544 **ASYNCHRONOUS EVENTS**
- 107545 Default.
- 107546 **STDOUT**
- 107547 Not used.
- 107548 **STDERR**
- 107549 The standard error shall be used only for diagnostic messages.
- 107550 **OUTPUT FILES**
- 107551 None.
- 107552 **EXTENDED DESCRIPTION**
- 107553 None.
- 107554 **EXIT STATUS**
- 107555 The following exit values shall be returned:
- 107556 0 Each directory entry specified by a *dir* operand was removed successfully.
- 107557 >0 An error occurred.
- 107558 **CONSEQUENCES OF ERRORS**
- 107559 Default.
- 107560 **APPLICATION USAGE**
- 107561 The definition of an empty directory is one that contains, at most, directory entries for dot and dot-dot.
- 107562
- 107563 **EXAMPLES**
- 107564 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty
- 107565 except it contains a directory **c**:
- 107566 `rmdir -p a/b/c`
- 107567 removes all three directories.
- 107568 **RATIONALE**
- 107569 On historical System V systems, the `-p` option also caused a message to be written to the
- 107570 standard output. The message indicated whether the whole path was removed or whether part
- 107571 of the path remained for some reason. The **STDERR** section requires this diagnostic when the
- 107572 entire path specified by a *dir* operand is not removed, but does not allow the status message
- 107573 reporting success to be written as a diagnostic.
- 107574 The *rmdir* utility on System V also included a `-s` option that suppressed the informational
- 107575 message output by the `-p` option. This option has been omitted because the informational
- 107576 message is not specified by this volume of POSIX.1-2017.
- 107577 **FUTURE DIRECTIONS**
- 107578 None.
- 107579 **SEE ALSO**
- 107580 *rm*
- 107581 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 107582 XSH *remove()*, *rmdir()*, *unlink()*

107583 **CHANGE HISTORY**

107584 First released in Issue 2.

107585 **Issue 6**

107586 The normative text is reworded to avoid use of the term “must” for application requirements.

107587 **NAME**107588 sact — print current SCCS file-editing activity (**DEVELOPMENT**)107589 **SYNOPSIS**107590 XSI `sact file...`107591 **DESCRIPTION**

107592 The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a
 107593 list to standard output. This situation occurs when *get -e* has been executed previously without
 107594 a subsequent execution of *delta*, *unget*, or *sccs unedit*.

107595 **OPTIONS**

107596 None.

107597 **OPERANDS**

107598 The following operand shall be supported:

107599 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact*
 107600 utility shall behave as though each file in the directory were specified as a named
 107601 file, except that non-SCCS files (last component of the pathname does not begin
 107602 with **s**.) and unreadable files shall be silently ignored.

107603 If exactly one *file* operand appears, and it is '-', the standard input shall be read;
 107604 each line of the standard input shall be taken to be the name of an SCCS file to be
 107605 processed. Non-SCCS files and unreadable files shall be silently ignored.

107606 **STDIN**

107607 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each
 107608 line of the text file shall be interpreted as an SCCS pathname.

107609 **INPUT FILES**

107610 Any SCCS files interrogated are files of an unspecified format.

107611 **ENVIRONMENT VARIABLES**107612 The following environment variables shall affect the execution of *sact*:

107613 *LANG* Provide a default value for the internationalization variables that are unset or null.
 107614 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 107615 variables used to determine the values of locale categories.)

107616 *LC_ALL* If set to a non-empty string value, override the values of all the other
 107617 internationalization variables.

107618 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 107619 characters (for example, single-byte as opposed to multi-byte characters in
 107620 arguments and input files).

107621 *LC_MESSAGES*

107622 Determine the locale that should be used to affect the format and contents of
 107623 diagnostic messages written to standard error.

107624 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

107625 **ASYNCHRONOUS EVENTS**

107626 Default.

107627 STDOUT

107628 The output for each named file shall consist of a line in the following format:

107629 "%sΔ%sΔ%sΔ%sΔ\n", <SID>, <new SID>, <login>, <date>, <time>

107630 <SID> Specifies the SID of a delta that currently exists in the SCCS file to which changes
107631 are made to make the new delta.

107632 <new SID> Specifies the SID for the new delta to be created.

107633 <login> Contains the login name of the user who makes the delta (that is, who executed a
107634 *get* for editing).

107635 <date> Contains the date that *get -e* was executed, in the format used by the *prs :D:* data
107636 keyword.

107637 <time> Contains the time that *get -e* was executed, in the format used by the *prs :T:* data
107638 keyword.

107639 If there is more than one named file or if a directory or standard input is named, each pathname
107640 shall be written before each of the preceding lines:

107641 "\n%s:\n", <pathname>

107642 STDERR

107643 The standard error shall be used only for optional informative messages concerning SCCS files
107644 with no impending deltas, and for diagnostic messages.

107645 OUTPUT FILES

107646 None.

107647 EXTENDED DESCRIPTION

107648 None.

107649 EXIT STATUS

107650 The following exit values shall be returned:

107651 0 Successful completion.

107652 >0 An error occurred.

107653 CONSEQUENCES OF ERRORS

107654 Default.

107655 APPLICATION USAGE

107656 None.

107657 EXAMPLES

107658 None.

107659 RATIONALE

107660 None.

107661 FUTURE DIRECTIONS

107662 None.

107663 SEE ALSO

107664 *delta, get, sccs, unget*

107665 XBD Chapter 8 (on page 173)

107666 **CHANGE HISTORY**
107667 First released in Issue 2.

107668 **NAME**107669 `sccs` — front end for the SCCS subsystem (**DEVELOPMENT**)107670 **SYNOPSIS**107671 XSI `sccs [-r] [-d path] [-p path] command [options...] [operands...]`107672 **DESCRIPTION**107673 The `sccs` utility is a front end to the SCCS programs. It also includes the capability to run set-user-id to another user to provide additional protection.
107674107675 The `sccs` utility shall invoke the specified *command* with the specified *options* and *operands*. By
107676 default, each of the *operands* shall be modified by prefixing it with the string "SCCS/s.". 107677 The *command* can be the name of one of the SCCS utilities in this volume of POSIX.1-2017 (*admin*,
107678 *delta*, *get*, *prs*, *rmdel*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the
107679 EXTENDED DESCRIPTION section.107680 **OPTIONS**107681 The `sccs` utility shall conform to XBD [Section 12.2](#) (on page 216), except that *options* operands are
107682 actually options to be passed to the utility named by *command*. When the portion of the
107683 command:107684 `command [options ...] [operands ...]`107685 is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax
107686 Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the
107687 Guidelines to the extent indicated by their individual OPTIONS sections.107688 The following options shall be supported preceding the *command* operand:107689 **-d path** A pathname of a directory to be used as a root directory for the SCCS files. The
107690 default shall be the current directory. The **-d** option shall take precedence over the
107691 *PROJECTDIR* variable. See **-p**.107692 **-p path** A pathname of a directory in which the SCCS files are located. The default shall be
107693 the **SCCS** directory.107694 The **-p** option differs from the **-d** option in that the **-d** option-argument shall be
107695 prefixed to the entire pathname and the **-p** option-argument shall be inserted
107696 before the final component of the pathname. For example:107697 `sccs -d /x -p y get a/b`

107698 converts to:

107699 `get /x/a/y/s.b`

107700 This allows the creation of aliases such as:

107701 `alias syssccs="sccs -d /usr/src"`

107702 which is used as:

107703 `syssccs get cmd/who.c`107704 **-r** Invoke *command* with the real user ID of the process, not any effective user ID that
107705 the `sccs` utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmdel*,
107706 and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to
107707 change the authorizations. These commands are always run as the real user.

107708 **OPERANDS**

107709 The following operands shall be supported:

107710 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the
107711 EXTENDED DESCRIPTION section.

107712 *options* An option or option-argument to be passed to *command*.

107713 *operands* An operand to be passed to *command*.

107714 **STDIN**

107715 See the utility description for the specified *command*.

107716 **INPUT FILES**

107717 See the utility description for the specified *command*.

107718 **ENVIRONMENT VARIABLES**

107719 The following environment variables shall affect the execution of *sccs*:

107720 *LANG* Provide a default value for the internationalization variables that are unset or null.
107721 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
107722 variables used to determine the values of locale categories.)

107723 *LC_ALL* If set to a non-empty string value, override the values of all the other
107724 internationalization variables.

107725 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
107726 characters (for example, single-byte as opposed to multi-byte characters in
107727 arguments and input files).

107728 *LC_MESSAGES*

107729 Determine the locale that should be used to affect the format and contents of
107730 diagnostic messages written to standard error.

107731 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

107732 *PROJECTDIR*

107733 Provide a default value for the *-d path* option. If the value of *PROJECTDIR* begins
107734 with a <slash>, it shall be considered an absolute pathname; otherwise, the value
107735 of *PROJECTDIR* is treated as a user name and that user's initial working directory
107736 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it
107737 shall be used. Otherwise, the value shall be used as a relative pathname.

107738 Additional environment variable effects may be found in the utility description for the specified
107739 *command*.

107740 **ASYNCHRONOUS EVENTS**

107741 Default.

107742 **STDOUT**

107743 See the utility description for the specified *command*.

107744 **STDERR**

107745 See the utility description for the specified *command*.

107746 **OUTPUT FILES**

107747 See the utility description for the specified *command*.

107748 EXTENDED DESCRIPTION

107749 The following pseudo-utilities shall be supported as *command* operands. All options referred to
 107750 in the following list are values given in the *options* operands following *command*.

107751 **check** Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a
 107752 non-zero exit status shall be returned if anything is being edited. The intent is to have
 107753 this included in an ``install'' entry in a makefile to ensure that everything is included
 107754 into the SCCS file before a version is installed.

107755 **clean** Remove everything from the current directory that can be recreated from SCCS files,
 107756 but do not remove any files being edited. If the **-b** option is given, branches shall be
 107757 ignored in the determination of whether they are being edited; this is dangerous if
 107758 branches are kept in the same directory.

107759 **create** Create an SCCS file, taking the initial contents from the file of the same name. Any
 107760 options to *admin* are accepted. If the creation is successful, the original files shall be
 107761 renamed by prefixing the basenames with a comma. These renamed files should be
 107762 removed after it has been verified that the SCCS files have been created successfully.

107763 **delget** Perform a *delta* on the named files and then *get* new versions. The new versions shall
 107764 have ID keywords expanded and shall not be editable. Any **-m**, **-p**, **-r**, **-s**, and **-y**
 107765 options shall be passed to *delta*, and any **-b**, **-c**, **-e**, **-i**, **-k**, **-l**, **-s**, and **-x** options shall be
 107766 passed to *get*.

107767 **deledit** Equivalent to **delget**, except that the *get* phase shall include the **-e** option. This option is
 107768 useful for making a checkpoint of the current editing phase. The same options shall be
 107769 passed to *delta* as described above, and all the options listed for *get* above except **-e**
 107770 shall be passed to **edit**.

107771 **diffs** Write a difference listing between the current version of the files checked out for editing
 107772 and the versions in SCCS format. Any **-r**, **-c**, **-i**, **-x**, and **-t** options shall be passed to
 107773 *get*; any **-l**, **-s**, **-e**, **-f**, **-h**, and **-b** options shall be passed to *diff*. A **-C** option shall be
 107774 passed to *diff* as **-c**.

107775 **edit** Equivalent to *get -e*.

107776 **fix** Remove the named delta, but leave a copy of the delta with the changes that were in it.
 107777 It is useful for fixing small compiler bugs, and so on. The application shall ensure that it
 107778 is followed by a **-r** *SID* option. Since **fix** does not leave audit trails, it should be used
 107779 carefully.

107780 **info** Write a listing of all files being edited. If the **-b** option is given, branches (that is, SIDs
 107781 with two or fewer components) shall be ignored. If a **-u** *user* option is given, then only
 107782 files being edited by the named user shall be listed. A **-U** option shall be equivalent to
 107783 **-u**<*current user*>.

107784 **print** Write out verbose information about the named files, equivalent to *sccs prs*.

107785 **tell** Write a <newline>-separated list of the files being edited to standard output. Takes the
 107786 **-b**, **-u**, and **-U** options like **info** and **check**.

107787 **unedit** This is the opposite of an **edit** or a *get -e*. It should be used with caution, since any
 107788 changes made since the *get* are lost.

107789 **EXIT STATUS**

107790 The following exit values shall be returned:

107791 0 Successful completion.

107792 >0 An error occurred.

107793 **CONSEQUENCES OF ERRORS**

107794 Default.

107795 **APPLICATION USAGE**

107796 Many of the SCCS utilities take directory names as operands as well as specific filenames. The
 107797 pseudo-utilities supported by *sccs* are not described as having this capability, but are not
 107798 prohibited from doing so.

107799 **EXAMPLES**

107800 1. To get a file for editing, edit it and produce a new delta:

107801 `sccs get -e file.c`107802 `ex file.c`107803 `sccs delta file.c`

107804 2. To get a file from another directory:

107805 `sccs -p /usr/src/sccs/s. get cc.c`

107806 or:

107807 `sccs get /usr/src/sccs/s.cc.c`

107808 3. To make a delta of a large number of files in the current directory:

107809 `sccs delta *.c`

107810 4. To get a list of files being edited that are not on branches:

107811 `sccs info -b`

107812 5. To delta everything being edited by the current user:

107813 `sccs delta $(sccs tell -U)`

107814 6. In a makefile, to get source files from an SCCS file if it does not already exist:

107815 `SRCS = <list of source files>`107816 `$(SRCS):`107817 `sccs get $(REL) $@`107818 **RATIONALE**

107819 *sccs* and its associated utilities are part of the XSI Development Utilities option within the XSI
 107820 option.

107821 SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement
 107822 tracking tool. When a file is put under SCCS, the source code control system maintains the file
 107823 and, when changes are made, identifies and stores them in the file with the original source code
 107824 and/or documentation. As other changes are made, they too are identified and retained in the
 107825 file.

107826 Retrieval of the original and any set of changes is possible. Any version of the file as it develops
 107827 can be reconstructed for inspection or additional modification. History data can be stored with
 107828 each version, documenting why the changes were made, who made them, and when they were
 107829 made.

107830 **FUTURE DIRECTIONS**

107831 None.

107832 **SEE ALSO**107833 *admin, delta, get, make, prs, rmdel, sact, unget, val, what*107834 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)107835 **CHANGE HISTORY**

107836 First released in Issue 4.

107837 **Issue 6**107838 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from
107839 ``otherwise, the home directory of a user of that name is examined'' to ``otherwise, the value of
107840 *PROJECTDIR* is treated as a user name and that user's initial working directory is examined''.

107841 The normative text is reworded to avoid use of the term ``must'' for application requirements.

107842 **Issue 7**

107843 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

107844 **NAME**

107845 sed — stream editor

107846 **SYNOPSIS**107847 sed [-n] *script* [*file...*]107848 sed [-n] -e *script* [-e *script*]... [-f *script_file*]... [*file...*]107849 sed [-n] [-e *script*]... -f *script_file* [-f *script_file*]... [*file...*]107850 **DESCRIPTION**

107851 The *sed* utility is a stream editor that shall read one or more text files, make editing changes
 107852 according to a script of editing commands, and write the results to standard output. The script
 107853 shall be obtained from either the *script* operand string or a combination of the option-arguments
 107854 from the -e *script* and -f *script_file* options.

107855 **OPTIONS**

107856 The *sed* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the order of
 107857 presentation of the -e and -f options is significant.

107858 The following options shall be supported:

107859 -e *script* Add the editing commands specified by the *script* option-argument to the end of
 107860 the script of editing commands.

107861 -f *script_file* Add the editing commands in the file *script_file* to the end of the script of editing
 107862 commands.

107863 -n Suppress the default output (in which each line, after it is examined for editing, is
 107864 written to standard output). Only lines explicitly selected for output are written.

107865 If any -e or -f options are specified, the script of editing commands shall initially be empty. The
 107866 commands specified by each -e or -f option shall be added to the script in the order specified.
 107867 When each addition is made, if the previous addition (if any) was from a -e option, a <newline>
 107868 shall be inserted before the new addition. The resulting script shall have the same properties as
 107869 the *script* operand, described in the OPERANDS section.

107870 **OPERANDS**

107871 The following operands shall be supported:

107872 *file* A pathname of a file whose contents are read and edited. If multiple *file* operands
 107873 are specified, the named files shall be read in the order specified and the
 107874 concatenation shall be edited. If no *file* operands are specified, the standard input
 107875 shall be used.

107876 *script* A string to be used as the script of editing commands. The application shall not
 107877 present a *script* that violates the restrictions of a text file except that the final
 107878 character need not be a <newline>.

107879 **STDIN**

107880 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
 107881 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
 107882 the standard input shall not be used. See the INPUT FILES section.

107883 **INPUT FILES**

107884 The input files shall be text files. The *script_files* named by the -f option shall consist of editing
 107885 commands.

107886 **ENVIRONMENT VARIABLES**

107887 The following environment variables shall affect the execution of *sed*:

107888 *LANG* Provide a default value for the internationalization variables that are unset or null.
 107889 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 107890 variables used to determine the values of locale categories.)

107891 *LC_ALL* If set to a non-empty string value, override the values of all the other
 107892 internationalization variables.

107893 *LC_COLLATE*

107894 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 107895 character collating elements within regular expressions.

107896 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 107897 characters (for example, single-byte as opposed to multi-byte characters in
 107898 arguments and input files), and the behavior of character classes within regular
 107899 expressions.

107900 *LC_MESSAGES*

107901 Determine the locale that should be used to affect the format and contents of
 107902 diagnostic messages written to standard error.

107903 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

107904 **ASYNCHRONOUS EVENTS**

107905 Default.

107906 **STDOUT**

107907 The input files shall be written to standard output, with the editing commands specified in the
 107908 script applied. If the *-n* option is specified, only those input lines selected by the script shall be
 107909 written to standard output.

107910 **STDERR**

107911 The standard error shall be used only for diagnostic and warning messages.

107912 **OUTPUT FILES**

107913 The output files shall be text files whose formats are dependent on the editing commands given.

107914 **EXTENDED DESCRIPTION**

107915 The *script* shall consist of editing commands of the following form:

107916 `[address[, address]] function`

107917 where *function* represents a single-character command verb from the list in [Editing Commands](#)
 107918 [in sed](#) (on page 3218), followed by any applicable arguments.

107919 The command can be preceded by <blank> characters and/or <semicolon> characters. The
 107920 function can be preceded by <blank> characters. These optional characters shall have no effect.

107921 In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>
 107922 character, into the pattern space. Reading from input shall be skipped if a <newline> was in the
 107923 pattern space prior to a **D** command ending the previous cycle. The *sed* utility shall then apply in
 107924 sequence all commands whose addresses select that pattern space, until a command starts the
 107925 next cycle or quits. If no commands explicitly started a new cycle, then at the end of the script
 107926 the pattern space shall be copied to standard output (except when *-n* is specified) and the
 107927 pattern space shall be deleted. Whenever the pattern space is written to standard output or a
 107928 named file, *sed* shall immediately follow it with a <newline>.

107929 Some of the editing commands use a hold space to save all or part of the pattern space for

107930 subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

107931 **Addresses in sed**

107932 An address is either a decimal number that counts input lines cumulatively across files, a '\$'
107933 character that addresses the last line of input, or a context address (which consists of a BRE, as
107934 described in [Regular Expressions in sed](#), preceded and followed by a delimiter, usually a
107935 <slash>).

107936 An editing command with no addresses shall select every pattern space.

107937 An editing command with one address shall select each pattern space that matches the address.

107938 An editing command with two addresses shall select the inclusive range from the first pattern
107939 space that matches the first address through the next pattern space that matches the second. (If
107940 the second address is a number less than or equal to the line number first selected, only one line
107941 shall be selected.) Starting at the first line following the selected range, *sed* shall look again for
107942 the first address. Thereafter, the process shall be repeated. Omitting either or both of the address
107943 components in the following form produces undefined results:

107944 `[address[,address]]`

107945 **Regular Expressions in sed**

107946 The *sed* utility shall support the BREs described in XBD [Section 9.3](#) (on page 183), with the
107947 following additions:

107948 In a context address, the construction "`\cBREc`", where *c* is any character other than
107949 <backslash> or <newline>, shall be identical to "`/BRE/`". If the character designated by *c*
107950 appears following a <backslash>, then it shall be considered to be that literal character,
107951 which shall not terminate the BRE. For example, in the context address "`\xabc\xdefx`",
107952 the second *x* stands for itself, so that the BRE is "`abcxdef`".

107953 The escape sequence '`\n`' shall match a <newline> embedded in the pattern space. A
107954 literal <newline> shall not be used in the BRE of a context address or in the substitute
107955 function.

107956 If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in
107957 the last command applied (either as an address or as part of a substitute command) was
107958 specified.

107959 **Editing Commands in sed**

107960 In the following list of editing commands, the maximum number of permissible addresses for
107961 each function is indicated by `[0addr]`, `[1addr]`, or `[2addr]`, representing zero, one, or two
107962 addresses.

107963 The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall
107964 be preceded by a <backslash>. Other <backslash> characters in text shall be removed, and the
107965 following character shall be treated literally.

107966 The **r** and **w** command verbs, and the *w* flag to the **s** command, take an *rfile* (or *wfile*) parameter,
107967 separated from the command verb letter or flag by one or more <blank> characters;
107968 implementations may allow zero separation as an extension.

107969 The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be
107970 created before processing begins. Implementations shall support at least ten *wfile* arguments in
107971 the script; the actual number (greater than or equal to 10) that is supported by the

107972 implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially
 107973 created, if it does not exist, or shall replace the contents of an existing file.

107974 The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following
 107975 synopses indicate which arguments shall be separated from the command verbs by a single
 107976 <space>.

107977 The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and
 107978 the contents of the file specified for the **r** command, shall be written to standard output just
 107979 before the next attempt to fetch a line of input when executing the **N** or **n** commands, or when
 107980 reaching the end of the script. If written when reaching the end of the script, and the **-n** option
 107981 was not specified, the text shall be written after copying the pattern space to standard output.
 107982 The contents of the file specified for the **r** command shall be as of the time the output is written,
 107983 not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r**
 107984 commands were applied to the input.

107985 Editing commands other than {...}, **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a <semicolon>,
 107986 optional <blank> characters, and another editing command. However, when an **s** editing
 107987 command is used with the *w* flag, following it with another command in this manner produces
 107988 undefined results.

107989 A function can be preceded by a **!** character, in which case the function shall be applied if the
 107990 addresses do not select the pattern space. Zero or more <blank> characters shall be accepted
 107991 before the **!** character. It is unspecified whether <blank> characters can follow the **!**
 107992 character, and conforming applications shall not follow the **!** character with <blank>
 107993 characters.

107994 If a *label* argument (to a **b**, **t**, or **:** command) contains characters outside of the portable filename
 107995 character set, or if a *label* is longer than 8 bytes, the behavior is unspecified. The implementation
 107996 shall support *label* arguments recognized as unique up to at least 8 bytes; the actual length
 107997 (greater than or equal to 8) supported by the implementation is unspecified. It is unspecified
 107998 whether exceeding the maximum supported label length causes an error or a silent truncation.

107999 [2addr] {editing command
 108000 editing command

108001 ...

108002 } Execute a list of *sed* editing commands only when the pattern space is selected. The
 108003 list of *sed* editing commands shall be surrounded by braces. The braces can be
 108004 preceded or followed by <blank> characters. The <right-brace> shall be preceded
 108005 by a <newline> or <semicolon> (before any optional <blank> characters preceding
 108006 the <right-brace>).

108007 Each command in the list of commands shall be terminated by a <newline>
 108008 character, or by a <semicolon> character if permitted when the command is used
 108009 outside the braces. The editing commands can be preceded by <blank> characters,
 108010 but shall not be followed by <blank> characters.

108011 [1addr]a\
 108012 text

Write text to standard output as described previously.

108013 [2addr]b [label]

108014 Branch to the **:** command verb bearing the *label* argument. If *label* is not specified,
 108015 branch to the end of the script.

108016 [2addr]c\
 108017 text

Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range,
 108018 place *text* on the output and start the next cycle.

108019	[2addr]d	Delete the pattern space and start the next cycle.
108020	[2addr]D	If the pattern space contains no <newline>, delete the pattern space and start a normal new cycle as if the d command was issued. Otherwise, delete the initial segment of the pattern space through the first <newline>, and start the next cycle with the resultant pattern space and without reading any new input.
108021		
108022		
108023		
108024	[2addr]g	
108025	[2addr]G	Append to the pattern space a <newline> followed by the contents of the hold space.
108026		
108027	[2addr]h	Replace the contents of the hold space with the contents of the pattern space.
108028	[2addr]H	Append to the hold space a <newline> followed by the contents of the pattern space.
108029		
108030	[1addr]i\	Write <i>text</i> to standard output.
108031	<i>text</i>	
108032	[2addr]l	(The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in XBD Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-printable characters not in that table shall be written as one three-digit octal number (with a preceding <backslash>) for each byte in the character (most significant byte first).
108033		
108034		
108035		
108036		
108037		Long lines shall be folded, with the point of folding indicated by writing a <backslash> followed by a <newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a '\$'.
108038		Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input, less its terminating <newline>.
108039		
108040		
108041		
108042	[2addr]n	
108043		If no next line of input is available, the n command verb shall branch to the end of the script and quit without starting a new cycle.
108044		
108045		Append the next line of input, less its terminating <newline>, to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.
108046		
108047	[2addr]N	
108048		
108049		
108050		If no next line of input is available, the N command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output.
108051		
108052		
108053	[2addr]p	Write the pattern space to standard output.
108054	[2addr]P	Write the pattern space, up to the first <newline>, to standard output.
108055	[1addr]q	Branch to the end of the script and quit without starting a new cycle.
108056	[1addr]r <i>rfile</i>	Copy the contents of <i>rfile</i> to standard output as described previously. If <i>rfile</i> does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.
108057		
108058		
108059	[2addr]s/BRE/replacement/flags	Substitute the replacement string for instances of the BRE in the pattern space. Any character other than <backslash> or <newline> can be used instead of a <slash> to
108060		
108061		

108062 delimit the BRE and the replacement. Within the BRE and the replacement, the
108063 BRE delimiter itself can be used as a literal character if it is preceded by a
108064 <backslash>.

108065 The replacement string shall be scanned from beginning to end. An <ampersand>
108066 ('&') appearing in the replacement shall be replaced by the string matching the
108067 BRE. The special meaning of '&' in this context can be suppressed by preceding it
108068 by a <backslash>. The characters "\n", where *n* is a digit, shall be replaced by the
108069 text matched by the corresponding back-reference expression. If the corresponding
108070 back-reference expression does not match, then the characters "\n" shall be
108071 replaced by the empty string. The special meaning of "\n" where *n* is a digit in
108072 this context, can be suppressed by preceding it by a <backslash>. For each other
108073 <backslash> encountered, the following character shall lose its special meaning (if
108074 any).

108075 A line can be split by substituting a <newline> into it. The application shall escape
108076 the <newline> in the replacement by preceding it by a <backslash>.

108077 The meaning of an unescaped <backslash> immediately followed by any character
108078 other than '&', <backslash>, a digit, <newline>, or the delimiter character used for
108079 this command, is unspecified.

108080 A substitution shall be considered to have been performed even if the replacement
108081 string is identical to the string that it replaces. Any <backslash> used to alter the
108082 default meaning of a subsequent character shall be discarded from the BRE or the
108083 replacement before evaluating the BRE or using the replacement.

108084 The value of *flags* shall be zero or more of:

108085 *n* Substitute for the *n*th occurrence only of the BRE found within the
108086 pattern space.

108087 **g** Globally substitute for all non-overlapping instances of the BRE
108088 rather than just the first one. If both **g** and *n* are specified, the results
108089 are unspecified.

108090 **p** Write the pattern space to standard output if a replacement was
108091 made.

108092 **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.
108093 A conforming application shall precede the *wfile* argument with one
108094 or more <blank> characters. If the **w** flag is not the last flag value
108095 given in a concatenation of multiple flag values, the results are
108096 undefined.

108097 [*2addr*]**t** [*label*]
108098 Test. Branch to the : command verb bearing the *label* if any substitutions have been
108099 made since the most recent reading of an input line or execution of a **t**. If *label* is
108100 not specified, branch to the end of the script.

108101 [*2addr*]**w** *wfile*
108102 Append (write) the pattern space to *wfile*.

108103 [*2addr*]**x** Exchange the contents of the pattern and hold spaces.

108104 [*2addr*]**y**/*string1*/*string2*/
108105 Replace all occurrences of characters in *string1* with the corresponding characters
108106 in *string2*. If a <backslash> followed by an 'n' appear in *string1* or *string2*, the two

108107 characters shall be handled as a single <newline>. If the number of characters in
 108108 *string1* and *string2* are not equal, or if any of the characters in *string1* appear more
 108109 than once, the results are undefined. Any character other than <backslash> or
 108110 <newline> can be used instead of <slash> to delimit the strings. If the delimiter is
 108111 not 'n', within *string1* and *string2*, the delimiter itself can be used as a literal
 108112 character if it is preceded by a <backslash>. If a <backslash> character is
 108113 immediately followed by a <backslash> character in *string1* or *string2*, the two
 108114 <backslash> characters shall be counted as a single literal <backslash> character.
 108115 The meaning of a <backslash> followed by any character that is not 'n', a
 108116 <backslash>, or the delimiter character is undefined.

108117 **[Oaddr]:label** Do nothing. This command bears a *label* to which the **b** and **t** commands branch.

108118 **[laddr]=** Write the following to standard output:

108119 `"%d\n", <current line number>`

108120 **[Oaddr]** Ignore this empty command.

108121 **[Oaddr]#** Ignore the '#' and the remainder of the line (treat them as a comment), with the
 108122 single exception that if the first two characters in the script are "#n", the default
 108123 output shall be suppressed; this shall be the equivalent of specifying **-n** on the
 108124 command line.

108125 EXIT STATUS

108126 The following exit values shall be returned:

108127 0 Successful completion.

108128 >0 An error occurred.

108129 CONSEQUENCES OF ERRORS

108130 Default.

108131 APPLICATION USAGE

108132 Regular expressions match entire strings, not just individual lines, but a <newline> is matched
 108133 by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression
 108134 in POSIX.1-2017. Also note that '\n' cannot be used to match a <newline> at the end of an
 108135 arbitrary input line; <newline> characters appear in the pattern space as a result of the **N** editing
 108136 command.

108137 When using *sed* to process pathnames, it is recommended that LC_ALL, or at least LC_CTYPE
 108138 and LC_COLLATE, are set to POSIX or C in the environment, since pathnames can contain byte
 108139 sequences that do not form valid characters in some locales, in which case the utility's behavior
 108140 would be undefined. In the POSIX locale each byte is a valid single-byte character, and therefore
 108141 this problem is avoided.

108142 EXAMPLES

108143 This *sed* script simulates the BSD *cat -s* command, squeezing excess empty lines from standard
 108144 input.

```
108145 sed -n '
108146 # Write non-empty lines.
108147 ./ {
108148     p
108149     d
108150 }
108151 # Write a single empty line, then look for more empty lines.
```

```

108152     /^$/      p
108153     # Get next line, discard the held <newline> (empty line),
108154     # and look for more empty lines.
108155     :Empty
108156     /^$/      {
108157         N
108158         s/.//
108159         b Empty
108160     }
108161     # Write the non-empty line before going back to search
108162     # for the first in a set of empty lines.
108163     p
108164     '

```

108165 The following *sed* command is a much simpler method of squeezing empty lines, although it is
 108166 not quite the same as *cat -s* since it removes any initial empty lines:

```
108167 sed -n '/./,/^$/p'
```

108168 RATIONALE

108169 This volume of POSIX.1-2017 requires implementations to support at least ten distinct *wfiles*,
 108170 matching historical practice on many implementations. Implementations are encouraged to
 108171 support more, but conforming applications should not exceed this limit.

108172 The exit status codes specified here are different from those in System V. System V returns 2 for
 108173 garbled *sed* commands, but returns zero with its usage message or if the input file could not be
 108174 opened. The standard developers considered this to be a bug.

108175 The manner in which the *l* command writes non-printable characters was changed to avoid the
 108176 historical backspace-overstrike method, and other requirements to achieve unambiguous output
 108177 were added. See the RATIONALE for *ed* for details of the format chosen, which is the same as
 108178 that chosen for *sed*.

108179 This volume of POSIX.1-2017 requires implementations to provide pattern and hold spaces of at
 108180 least 8 192 bytes, larger than the 4 000 bytes spaces used by some historical implementations, but
 108181 less than the 20 480 bytes limit used in an early proposal. Implementations are encouraged to
 108182 allocate dynamically larger pattern and hold spaces as needed.

108183 The requirements for acceptance of <blank> and <space> characters in command lines has been
 108184 made more explicit than in early proposals to describe clearly the historical practice and to
 108185 remove confusion about the phrase “protect initial blanks [*sic*] and tabs from the stripping that is
 108186 done on every script line” that appears in much of the historical documentation of the *sed* utility
 108187 description of text. (Not all implementations are known to have stripped <blank> characters
 108188 from text lines, although they all have allowed leading <blank> characters preceding the address
 108189 on a command line.)

108190 The treatment of '#' comments differs from the SVID which only allows a comment as the first
 108191 line of the script, but matches BSD-derived implementations. The comment character is treated
 108192 as a command, and it has the same properties in terms of being accepted with leading <blank>
 108193 characters; the BSD implementation has historically supported this.

108194 Early proposals required that a *script_file* have at least one non-comment line. Some historical
 108195 implementations have behaved in unexpected ways if this were not the case. The standard
 108196 developers considered that this was incorrect behavior and that application developers should
 108197 not have to avoid this feature. A correct implementation of this volume of POSIX.1-2017 shall
 108198 permit *script_files* that consist only of comment lines.

108199 Early proposals indicated that if `-e` and `-f` options were intermixed, all `-e` options were
 108200 processed before any `-f` options. This has been changed to process them in the order presented
 108201 because it matches historical practice and is more intuitive.

108202 The treatment of the `p` flag to the `s` command differs between System V and BSD-based systems
 108203 when the default output is suppressed. In the two examples:

```
108204 echo a | sed 's/a/A/p'
108205 echo a | sed -n 's/a/A/p'
```

108206 this volume of POSIX.1-2017, BSD, System V documentation, and the SVID indicate that the first
 108207 example should write two lines with **A**, whereas the second should write one. Some System V
 108208 systems write the **A** only once in both examples because the `p` flag is ignored if the `-n` option is
 108209 not specified.

108210 This is a case of a diametrical difference between systems that could not be reconciled through
 108211 the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V
 108212 documentation behavior was adopted for this volume of POSIX.1-2017 because:

108213 No known documentation for any historic system describes the interaction between the `p`
 108214 flag and the `-n` option.

108215 The selected behavior is more correct as there is no technical justification for any
 108216 interaction between the `p` flag and the `-n` option. A relationship between `-n` and the `p` flag
 108217 might imply that they are only used together, but this ignores valid scripts that interrupt
 108218 the cyclical nature of the processing through the use of the `D`, `d`, `q`, or branching
 108219 commands. Such scripts rely on the `p` suffix to write the pattern space because they do not
 108220 make use of the default output at the “bottom” of the script.

108221 Because the `-n` option makes the `p` flag unnecessary, any interaction would only be useful
 108222 if `sed` scripts were written to run both with and without the `-n` option. This is believed to
 108223 be unlikely. It is even more unlikely that programmers have coded the `p` flag expecting it to
 108224 be unnecessary. Because the interaction was not documented, the likelihood of a
 108225 programmer discovering the interaction and depending on it is further decreased.

108226 Finally, scripts that break under the specified behavior produce too much output instead of
 108227 too little, which is easier to diagnose and correct.

108228 The form of the substitute command that uses the `n` suffix was limited to the first 512 matches in
 108229 an early proposal. This limit has been removed because there is no reason an editor processing
 108230 lines of `{LINE_MAX}` length should have this restriction. The command `s/a/A/2047` should be
 108231 able to substitute the 2047th occurrence of `a` on a line.

108232 The `b`, `t`, and `:` commands are documented to ignore leading white space, but no mention is
 108233 made of trailing white space. Historical implementations of `sed` assigned different locations to
 108234 the labels `'x'` and `"x "`. This is not useful, and leads to subtle programming errors, but it is
 108235 historical practice, and changing it could theoretically break working scripts. Implementors are
 108236 encouraged to provide warning messages about labels that are never referenced by a `b` or `t`
 108237 command, jumps to labels that do not exist, and label arguments that are subject to truncation.

108238 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
 108239 but this has been modified in this version.

108240 FUTURE DIRECTIONS

108241 None.

108242 **SEE ALSO**108243 *awk, ed, grep*108244 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 9.3](#) (on page 183), [Section 12.2](#) (on
108245 page 216)108246 **CHANGE HISTORY**

108247 First released in Issue 2.

108248 **Issue 5**

108249 The FUTURE DIRECTIONS section is added.

108250 **Issue 6**108251 The following new requirements on POSIX implementations derive from alignment with the
108252 Single UNIX Specification:108253 Implementations are required to support at least ten *wfile* arguments in an editing
108254 command.

108255 The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

108256 IEEE PASC Interpretation 1003.2 #190 is applied.

108257 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the
108258 <backslash>-escape sequences in a replacement string for a BRE.108259 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/28 is applied, removing text describing
108260 behavior on systems with bytes consisting of more than eight bits.108261 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/29 is applied, making an editorial
108262 correction within the Editing Commands in *sed* section.108263 **Issue 7**

108264 Austin Group Interpretations 1003.1-2001 #006, #036, and #092 are applied.

108265 SD5-XCU-ERN-97 and SD5-XCU-ERN-123 are applied, updating the SYNOPSIS.

108266 A second example is added.

108267 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0133 [262], XCU/TC1-2008/0134
108268 [282,431], XCU/TC1-2008/0135 [269], and XCU/TC1-2008/0136 [282,431] are applied.108269 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0166 [945], XCU/TC2-2008/0167
108270 [944], XCU/TC2-2008/0168 [945], XCU/TC2-2008/0169 [944], XCU/TC2-2008/0170 [945],
108271 XCU/TC2-2008/0171 [533], XCU/TC2-2008/0172 [663], XCU/TC2-2008/0173 [945], and
108272 XCU/TC2-2008/0174 [944] are applied.

108273 **NAME**

108274 sh *†*'shell, the standard command language interpreter

108275 **SYNOPSIS**

108276 sh [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...
 108277 [*command_file* [*argument*...]]

108278 sh -c [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...
 108279 *command_string* [*command_name* [*argument*...]]

108280 sh -s [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...
 108281 [*argument*...]

108282 **DESCRIPTION**

108283 The *sh* utility is a command language interpreter that shall execute commands read from a
 108284 command line string, the standard input, or a specified file. The application shall ensure that the
 108285 commands to be executed are expressed in the language described in [Chapter 2](#) (on page 2345).

108286 Pathname expansion shall not fail due to the size of a file.

108287 Shell input and output redirections have an implementation-defined offset maximum that is
 108288 established in the open file description.

108289 **OPTIONS**

108290 The *sh* utility shall conform to XBD [Section 12.2](#) (on page 216), with an extension for support of a
 108291 leading <plus-sign> ('+') as noted below.

108292 The *-a*, *-b*, *-C*, *-e*, *-f*, *-m*, *-n*, *-o option*, *-u*, *-v*, and *-x* options are described as part of the *set*
 108293 utility in [Section 2.14](#) (on page 2384). The option letters derived from the *set* special built-in shall
 108294 also be accepted with a leading <plus-sign> ('+') instead of a leading <hyphen-minus>
 108295 (meaning the reverse case of the option as described in this volume of POSIX.1-2017).

108296 The following additional options shall be supported:

108297 *-c* Read commands from the *command_string* operand. Set the value of special
 108298 parameter 0 (see [Section 2.5.2](#), on page 2350) from the value of the *command_name*
 108299 operand and the positional parameters (\$1, \$2, and so on) in sequence from the
 108300 remaining *argument* operands. No commands shall be read from the standard
 108301 input.

108302 *-i* Specify that the shell is *interactive*; see below. An implementation may treat
 108303 specifying the *-i* option as an error if the real user ID of the calling process does
 108304 not equal the effective user ID or if the real group ID does not equal the effective
 108305 group ID.

108306 *-s* Read commands from the standard input.

108307 If there are no operands and the *-c* option is not specified, the *-s* option shall be assumed.

108308 If the *-i* option is present, or if there are no operands and the shell's standard input and
 108309 standard error are attached to a terminal, the shell is considered to be *interactive*.

108310 **OPERANDS**

108311 The following operands shall be supported:

108312 *-* A single <hyphen-minus> shall be treated as the first operand and then ignored. If
 108313 both '-' and "--" are given as arguments, or if other operands precede the single
 108314 <hyphen-minus>, the results are undefined.

108315 *argument* The positional parameters (\$1, \$2, and so on) shall be set to *arguments*, if any.

108316 *command_file* The pathname of a file containing commands. If the pathname contains one or
108317 more <slash> characters, the implementation attempts to read that file; the file
108318 need not be executable. If the pathname does not contain a <slash> character:

108319 The implementation shall attempt to read that file from the current working
108320 directory; the file need not be executable.

108321 If the file is not in the current working directory, the implementation may
108322 perform a search for an executable file using the value of *PATH*, as described
108323 in [Section 2.9.1.1](#) (on page 2367).

108324 Special parameter 0 (see [Section 2.5.2](#), on page 2350) shall be set to the value of
108325 *command_file*. If *sh* is called using a synopsis form that omits *command_file*, special
108326 parameter 0 shall be set to the value of the first argument passed to *sh* from its
108327 parent (for example, *argv*[0] for a C program), which is normally a pathname used
108328 to execute the *sh* utility.

108329 *command_name* A string assigned to special parameter 0 when executing the commands in
108330 *command_string*. If *command_name* is not specified, special parameter 0 shall be set
108331 to the value of the first argument passed to *sh* from its parent (for example, *argv*[0]
108332 for a C program), which is normally a pathname used to execute the *sh* utility.
108333

108334 *command_string* A string that shall be interpreted by the shell as one or more commands, as if the
108335 string were the argument to the *system()* function defined in the System Interfaces
108336 volume of POSIX.1-2017. If the *command_string* operand is an empty string, *sh* shall
108337 exit with a zero exit status.
108338

108339 **STDIN**

108340 The standard input shall be used only if one of the following is true:

108341 The *-s* option is specified.

108342 The *-c* option is not specified and no operands are specified.

108343 The script executes one or more commands that require input from standard input (such as
108344 a *read* command that does not redirect its input).

108345 See the INPUT FILES section.

108346 When the shell is using standard input and it invokes a command that also uses standard input,
108347 the shell shall ensure that the standard input file pointer points directly after the command it has
108348 read when the command begins execution. It shall not read ahead in such a manner that any
108349 characters intended to be read by the invoked command are consumed by the shell (whether
108350 interpreted by the shell or not) or that characters that are not read by the invoked command are
108351 not seen by the shell. When the command expecting to read standard input is started
108352 asynchronously by an interactive shell, it is unspecified whether characters are read by the
108353 command or interpreted by the shell.

108354 If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh*
108355 shall enable blocking reads on standard input. This shall remain in effect when the command
108356 completes.

108357 **INPUT FILES**

108358 The input file shall be a text file, except that line lengths shall be unlimited. If the input file
 108359 consists solely of zero or more blank lines and comments, *sh* shall exit with a zero exit status.

108360 **ENVIRONMENT VARIABLES**

108361 The following environment variables shall affect the execution of *sh*:

108362 UP **ENV** This variable, when and only when an interactive shell is invoked, shall be
 108363 subjected to parameter expansion (see [Section 2.6.2](#), on page 2354) by the shell, and
 108364 the resulting value shall be used as a pathname of a file containing shell
 108365 commands to execute in the current environment. The file need not be executable.
 108366 If the expanded value of *ENV* is not an absolute pathname, the results are
 108367 unspecified. *ENV* shall be ignored if the real and effective user IDs or real and
 108368 effective group IDs of the process are different.

108369 UP **FCEDIT** This variable, when expanded by the shell, shall determine the default value for
 108370 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
 108371 used as the editor.

108372 UP **HISTFILE** Determine a pathname naming a command history file. If the *HISTFILE* variable is
 108373 not set, the shell may attempt to access or create a file **.sh_history** in the directory
 108374 referred to by the *HOME* environment variable. If the shell cannot obtain both read
 108375 and write access to, or create, the history file, it shall use an unspecified
 108376 mechanism that allows the history to operate properly. (References to history
 108377 *file* in this section shall be understood to mean this unspecified mechanism in
 108378 such cases.) An implementation may choose to access this variable only when
 108379 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
 108380 to retrieve entries from, or add entries to, the file, as the result of commands issued
 108381 by the user, the file named by the *ENV* variable, or implementation-defined system
 108382 start-up files. Implementations may choose to disable the history list mechanism
 108383 for users with appropriate privileges who do not set *HISTFILE*; the specific
 108384 circumstances under which this occurs are implementation-defined. If more than
 108385 one instance of the shell is using the same history file, it is unspecified how
 108386 updates to the history file from those shells interact. As entries are deleted from the
 108387 history file, they shall be deleted oldest first. It is unspecified when history file
 108388 entries are physically removed from the history file.

108389 **HISTSIZ** Determine a decimal number representing the limit to the number of previous
 108390 commands that are accessible. If this variable is unset, an unspecified default
 108391 greater than or equal to 128 shall be used. The maximum number of commands in
 108392 the history list is unspecified, but shall be at least 128. An implementation may
 108393 choose to access this variable only when initializing the history file, as described
 108394 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZ*
 108395 after the history file has been initialized are effective.

108396 UP **HOME** Determine the pathname of the user's home directory. The contents of *HOME* are
 108397 used in tilde expansion as described in [Section 2.6.1](#) (on page 2354).

108398 **LANG** Provide a default value for the internationalization variables that are unset or null.
 108399 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 108400 variables used to determine the values of locale categories.)

108401 **LC_ALL** If set to a non-empty string value, override the values of all the other
 108402 internationalization variables.

108403		<i>LC_COLLATE</i>	
108404			Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.
108405			
108406		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class alpha), and the behavior of character classes within pattern matching.
108407			
108408			
108409			
108410		<i>LC_MESSAGES</i>	
108411			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
108412			
108413	UP	<i>MAIL</i>	Determine a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to standard error prior to the writing of the next primary prompt string. Such check shall be performed only after the completion of the interval defined by the <i>MAILCHECK</i> variable after the last such check. The user shall be informed only if <i>MAIL</i> is set and <i>MAILPATH</i> is not set.
108414			
108415			
108416			
108417			
108418			
108419			
108420			
108421	UP	<i>MAILCHECK</i>	
108422			Establish a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt.
108423			
108424			
108425			
108426	UP	<i>MAILPATH</i>	Provide a list of pathnames and optional messages separated by <colon> characters. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for <i>MAIL</i> for descriptions of mail arrival and user informing.) Each pathname can be followed by ' <i>%</i> ' and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a ' <i>%</i> ' character in the pathname is preceded by a <backslash>, it shall be treated as a literal ' <i>%</i> ' in the pathname. The default message is unspecified.
108427			
108428			
108429			
108430			
108431			
108432			
108433			
108434			The <i>MAILPATH</i> environment variable takes precedence over the <i>MAIL</i> variable.
108435	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
108436		<i>PATH</i>	Establish a string formatted as described in XBD Chapter 8 (on page 173), used to effect command interpretation; see Section 2.9.1.1 (on page 2367).
108437			
108438		<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. Assignments to this variable may be ignored.
108439			
108440		ASYNCHRONOUS EVENTS	
108441			The <i>sh</i> utility shall take the standard action for all signals (see Section 1.4, on page 2336) with the following exceptions.
108442			
108443			If the shell is interactive, SIGINT signals received during command line editing shall be handled as described in the EXTENDED DESCRIPTION, and SIGINT signals received at other times shall be caught but no action performed.
108444			
108445			

108446 If the shell is interactive:

108447 SIGQUIT and SIGTERM signals shall be ignored.

108448 If the `-m` option is in effect, SIGTTIN, SIGTTOU, and SIGTSTP signals shall be ignored.

108449 If the `-m` option is not in effect, it is unspecified whether SIGTTIN, SIGTTOU, and
108450 SIGTSTP signals are ignored, set to the default action, or caught. If they are caught, the
108451 shell shall, in the signal-catching function, set the signal to the default action and raise the
108452 signal (after taking any appropriate steps, such as restoring terminal settings).

108453 The standard actions, and the actions described above for interactive shells, can be overridden
108454 by use of the `trap` special built-in utility (see `trap` (on page 2420) and [Section 2.11](#), on page 2381).

108455 **STDOUT**

108456 See the STDERR section.

108457 **STDERR**

108458 Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode),
108459 standard error shall be used only for diagnostic messages.

108460 **OUTPUT FILES**

108461 None.

108462 **EXTENDED DESCRIPTION**

108463 UP See [Chapter 2](#). The functionality described in the rest of the EXTENDED DESCRIPTION section
108464 shall be provided on implementations that support the User Portability Utilities option (and the
108465 rest of this section is not further shaded for this option).

108466 **Command History List**

108467 When the `sh` utility is being used interactively, it shall maintain a list of commands previously
108468 entered from the terminal in the file named by the `HISTFILE` environment variable. The type,
108469 size, and internal format of this file are unspecified. Multiple `sh` processes can share access to the
108470 file for a user, if file access permissions allow this; see the description of the `HISTFILE`
108471 environment variable.

108472 **Command Line Editing**

108473 When `sh` is being used interactively from a terminal, the current command and the command
108474 history (see `fc`) can be edited using *vi*-mode command line editing. This mode uses commands,
108475 described below, similar to a subset of those described in the *vi* utility. Implementations may
108476 offer other command line editing modes corresponding to other editing utilities.

108477 The command `set -o vi` shall enable *vi*-mode editing and place `sh` into *vi* insert mode (see
108478 [Command Line Editing \(vi-mode\)](#), on page 3231). This command also shall disable any other
108479 editing mode that the implementation may provide. The command `set +o vi` disables *vi*-mode
108480 editing.

108481 Certain block-mode terminals may be unable to support shell command line editing. If a
108482 terminal is unable to provide either edit mode, it need not be possible to `set -o vi` when using the
108483 shell on this terminal.

108484 In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the
108485 `stty` utility.

108486 **Command Line Editing (vi-mode)**

108487 In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations
 108488 which modify a line affect the edit line. The edit line is always the newest line in the command
 108489 history buffer.

108490 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

108491 When in insert mode, an entered character shall be inserted into the command line, except as
 108492 noted in [vi Line Editing Insert Mode](#). Upon entering *sh* and after termination of the previous
 108493 command, *sh* shall be in insert mode.

108494 Typing an escape character shall switch *sh* into command mode (see [vi Line Editing Command](#)
 108495 [Mode](#), on page 3232). In command mode, an entered character shall either invoke a defined
 108496 operation, be used as part of a multi-character operation, or be treated as an error. A character
 108497 that is not recognized as part of an editing command shall terminate any specific editing
 108498 command and shall alert the terminal. If *sh* receives a SIGINT signal in command mode
 108499 (whether generated by typing the *interrupt* character or by other means), it shall terminate
 108500 command line editing on the current command line, reissue the prompt on the next line of the
 108501 terminal, and reset the command history (see *fc*) so that the most recently executed command is
 108502 the previous command (that is, the command that was being edited when it was interrupted is
 108503 not re-entered into the history).

108504 In the following sections, the phrase “move the cursor to the beginning of the word” shall mean
 108505 “move the cursor to the first character of the current word” and the phrase “move the cursor to
 108506 the end of the word” shall mean “move the cursor to the last character of the current word”. The
 108507 phrase “beginning of the command line” indicates the point between the end of the prompt
 108508 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and
 108509 the first character of the command text.

108510 **vi Line Editing Insert Mode**

108511 While in insert mode, any character typed shall be inserted in the current command line, unless
 108512 it is from the following set.

108513 <newline> Execute the current command line. If the current command line is not empty, this
 108514 line shall be entered into the command history (see *fc*).

108515 *erase* Delete the character previous to the current cursor position and move the current
 108516 cursor position back one character. In insert mode, characters shall be erased from
 108517 both the screen and the buffer when backspacing.

108518 *interrupt* If *sh* receives a SIGINT signal in insert mode (whether generated by typing the
 108519 *interrupt* character or by other means), it shall terminate command line editing
 108520 with the same effects as described for interrupting command mode; see [Command](#)
 108521 [Line Editing \(vi-mode\)](#).

108522 *kill* Clear all the characters from the input line.

108523 <control>-V Insert the next character input, even if the character is otherwise a special insert
 108524 mode character.

108525 <control>-W Delete the characters from the one preceding the cursor to the preceding word
 108526 boundary. The word boundary in this case is the closer to the cursor of either the
 108527 beginning of the line or a character that is in neither the **blank** nor **punct** character
 108528 classification of the current locale.

108529 *end-of-file* Interpreted as the end of input in *sh*. This interpretation shall occur only at the
 108530 beginning of an input line. If *end-of-file* is entered other than at the beginning of the
 108531 line, the results are unspecified.

108532 <ESC> Place *sh* into command mode.

108533 **vi Line Editing Command Mode**

108534 In command mode for the command line editing feature, decimal digits not beginning with 0
 108535 that precede a command letter shall be remembered. Some commands use these decimal digits
 108536 as a count number that affects the operation.

108537 The term *motion command* represents one of the commands:

108538 <space> 0 b F l W ^ \$; E f T w | , B e h t

108539 If the current line is not the edit line, any command that modifies the current line shall cause the
 108540 content of the current line to replace the content of the edit line, and the current line shall
 108541 become the edit line. This replacement cannot be undone (see the **u** and **U** commands below).
 108542 The modification requested shall then be performed to the edit line. When the current line is the
 108543 edit line, the modification shall be done directly to the edit line.

108544 Any command that is preceded by *count* shall take a count (the numeric value of any preceding
 108545 decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat
 108546 by the number of times specified by the count. Also unless otherwise noted, a *count* that is out
 108547 of range is considered an error condition and shall alert the terminal, but neither the cursor
 108548 position, nor the command line, shall change.

108549 The terms *word* and *bigword* are used as defined in the *vi* description. The term *save buffer*
 108550 corresponds to the term *unnamed buffer* in *vi*.

108551 The following commands shall be recognized in command mode:

108552 <newline> Execute the current command line. If the current command line is not empty, this
 108553 line shall be entered into the command history (see *fc*).

108554 <control>-L Redraw the current command line. Position the cursor at the same location on the
 108555 redrawn line.

108556 # Insert the character '#' at the beginning of the current command line and treat the
 108557 resulting edit line as a comment. This line shall be entered into the command
 108558 history; see *fc*.

108559 = Display the possible shell word expansions (see [Section 2.6](#), on page 2353) of the
 108560 bigword at the current command line position.

108561 **Note:** This does not modify the content of the current line, and therefore does not cause
 108562 the current line to become the edit line.

108563 These expansions shall be displayed on subsequent terminal lines. If the bigword
 108564 contains none of the characters '?', '*', or '[', an <asterisk> ('*') shall be
 108565 implicitly assumed at the end. If any directories are matched, these expansions
 108566 shall have a '/' character appended. After the expansion, the line shall be
 108567 redrawn, the cursor repositioned at the current cursor position, and *sh* shall be
 108568 placed in command mode.

108569 \ Perform pathname expansion (see [Section 2.6.6](#), on page 2360) on the current
 108570 bigword, up to the largest set of characters that can be matched uniquely. If the
 108571 bigword contains none of the characters '?', '*', or '[', an <asterisk> ('*')
 108572 shall be implicitly assumed at the end. This maximal expansion then shall replace

108573 the original bigword in the command line, and the cursor shall be placed after this
 108574 expansion. If the resulting bigword completely and uniquely matches a directory, a
 108575 '/' character shall be inserted directly after the bigword. If some other file is
 108576 completely matched, a single <space> shall be inserted after the bigword. After
 108577 this operation, *sh* shall be placed in insert mode.

108578 * Perform pathname expansion on the current bigword and insert all expansions
 108579 into the command to replace the current bigword, with each expansion separated
 108580 by a single <space>. If at the end of the line, the current cursor position shall be
 108581 moved to the first column position following the expansions and *sh* shall be placed
 108582 in insert mode. Otherwise, the current cursor position shall be the last column
 108583 position of the first character after the expansions and *sh* shall be placed in insert
 108584 mode. If the current bigword contains none of the characters '?', '*', or '[',
 108585 before the operation, an <asterisk> ('*') shall be implicitly assumed at the end.

108586 @letter Insert the value of the alias named *_letter*. The symbol *letter* represents a single
 108587 alphabetic character from the portable character set; implementations may support
 108588 additional characters as an extension. If the alias *_letter* contains other editing
 108589 commands, these commands shall be performed as part of the insertion. If no alias
 108590 *_letter* is enabled, this command shall have no effect.

108591 [count]~ Convert, if the current character is a lowercase letter, to the equivalent uppercase
 108592 letter and *vice versa*, as prescribed by the current locale. The current cursor position
 108593 then shall be advanced by one character. If the cursor was positioned on the last
 108594 character of the line, the case conversion shall occur, but the cursor shall not
 108595 advance. If the '~' command is preceded by a *count*, that number of characters
 108596 shall be converted, and the cursor shall be advanced to the character position after
 108597 the last character converted. If the *count* is larger than the number of characters
 108598 after the cursor, this shall not be considered an error; the cursor shall advance to
 108599 the last character on the line.

108600 [count]. Repeat the most recent non-motion command, even if it was executed on an earlier
 108601 command line. If the previous command was preceded by a *count*, and no count is
 108602 given on the '.' command, the count from the previous command shall be
 108603 included as part of the repeated command. If the '.' command is preceded by a
 108604 *count*, this shall override any *count* argument to the previous command. The *count*
 108605 specified in the '.' command shall become the count for subsequent '.'
 108606 commands issued without a count.

108607 [number]v Invoke the *vi* editor to edit the current command line in a temporary file. When the
 108608 editor exits, the commands in the temporary file shall be executed and placed in
 108609 the command history. If a *number* is included, it specifies the command number in
 108610 the command history to be edited, rather than the current command line.

108611 [count]l (ell)
 108612 [count]<space>
 108613 Move the current cursor position to the next character position. If the cursor was
 108614 positioned on the last character of the line, the terminal shall be alerted and the
 108615 cursor shall not be advanced. If the *count* is larger than the number of characters
 108616 after the cursor, this shall not be considered an error; the cursor shall advance to
 108617 the last character on the line.

108618 [count]h Move the current cursor position to the *count*th (default 1) previous character
 108619 position. If the cursor was positioned on the first character of the line, the terminal
 108620 shall be alerted and the cursor shall not be moved. If the count is larger than the

108621		number of characters before the cursor, this shall not be considered an error; the
108622		cursor shall move to the first character on the line.
108623	[count]w	Move to the start of the next word. If the cursor was positioned on the last
108624		character of the line, the terminal shall be alerted and the cursor shall not be
108625		advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall
108626		not be considered an error; the cursor shall advance to the last character on the
108627		line.
108628	[count]W	Move to the start of the next bigword. If the cursor was positioned on the last
108629		character of the line, the terminal shall be alerted and the cursor shall not be
108630		advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this
108631		shall not be considered an error; the cursor shall advance to the last character on
108632		the line.
108633	[count]e	Move to the end of the current word. If at the end of a word, move to the end of
108634		the next word. If the cursor was positioned on the last character of the line, the
108635		terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger
108636		than the number of words after the cursor, this shall not be considered an error; the
108637		cursor shall advance to the last character on the line.
108638	[count]E	Move to the end of the current bigword. If at the end of a bigword, move to the
108639		end of the next bigword. If the cursor was positioned on the last character of the
108640		line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i>
108641		is larger than the number of bigwords after the cursor, this shall not be considered
108642		an error; the cursor shall advance to the last character on the line.
108643	[count]b	Move to the beginning of the current word. If at the beginning of a word, move to
108644		the beginning of the previous word. If the cursor was positioned on the first
108645		character of the line, the terminal shall be alerted and the cursor shall not be
108646		moved. If the <i>count</i> is larger than the number of words preceding the cursor, this
108647		shall not be considered an error; the cursor shall return to the first character on the
108648		line.
108649	[count]B	Move to the beginning of the current bigword. If at the beginning of a bigword,
108650		move to the beginning of the previous bigword. If the cursor was positioned on the
108651		first character of the line, the terminal shall be alerted and the cursor shall not be
108652		moved. If the <i>count</i> is larger than the number of bigwords preceding the cursor,
108653		this shall not be considered an error; the cursor shall return to the first character on
108654		the line.
108655	^	Move the current cursor position to the first character on the input line that is not a
108656		<blank>.
108657	\$	Move to the last character position on the current command line.
108658	0	(Zero.) Move to the first character position on the current command line.
108659	[count]l	Move to the <i>count</i> th character position on the current command line. If no number
108660		is specified, move to the first position. The first character position shall be
108661		numbered 1. If the count is larger than the number of characters on the line, this
108662		shall not be considered an error; the cursor shall be placed on the last character on
108663		the line.
108664	[count]fc	Move to the first occurrence of the character 'c' that occurs after the current
108665		cursor position. If the cursor was positioned on the last character of the line, the
108666		terminal shall be alerted and the cursor shall not be advanced. If the character 'c'

108667		does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
108668		
108669	[count]Fc	Move to the first occurrence of the character 'c' that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character 'c' does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
108670		
108671		
108672		
108673		
108674	[count]tc	Move to the character before the first occurrence of the character 'c' that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character 'c' does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
108675		
108676		
108677		
108678		
108679	[count]Tc	Move to the character after the first occurrence of the character 'c' that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character 'c' does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
108680		
108681		
108682		
108683		
108684	[count];	Repeat the most recent f , F , t , or T command. Any number argument on that previous command shall be ignored. Errors are those described for the repeated command.
108685		
108686		
108687	[count],	Repeat the most recent f , F , t , or T command. Any number argument on that previous command shall be ignored. However, reverse the direction of that command.
108688		
108689		
108690	a	Enter insert mode after the current cursor position. Characters that are entered shall be inserted before the next character.
108691		
108692	A	Enter insert mode after the end of the current command line.
108693	i	Enter insert mode at the current cursor position. Characters that are entered shall be inserted before the current character.
108694		
108695	I	Enter insert mode at the beginning of the current command line.
108696	R	Enter insert mode, replacing characters from the command line beginning at the current cursor position.
108697		
108698	[count]cmotion	
108699		Delete the characters between the current cursor position and the cursor position that would result from the specified motion command. Then enter insert mode before the first character following any deleted characters. If <i>count</i> is specified, it shall be applied to the motion command. A <i>count</i> shall be ignored for the following motion commands:
108700		
108701		
108702		
108703		
108704		0 ^ \$ c
108705		If the motion command is the character 'c', the current command line shall be cleared and insert mode shall be entered. If the motion command would move the current cursor position toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the motion command would move the current cursor position toward the end of the command line, the character under the current cursor position shall be deleted. If the <i>count</i> is larger than the number of characters between the current cursor position and the end of
108706		
108707		
108708		
108709		
108710		
108711		

108712 the command line toward which the motion command would move the cursor, this
 108713 shall not be considered an error; all of the remaining characters in the
 108714 aforementioned range shall be deleted and insert mode shall be entered. If the
 108715 motion command is invalid, the terminal shall be alerted, the cursor shall not be
 108716 moved, and no text shall be deleted.

108717 **C** Delete from the current character to the end of the line and enter insert mode at the
 108718 new end-of-line.

108719 **S** Clear the entire edit line and enter insert mode.

108720 **[count]rc** Replace the current character with the character 'c'. With a number *count*,
 108721 replace the current and the following *count*-1 characters. After this command, the
 108722 current cursor position shall be on the last character that was changed. If the *count*
 108723 is larger than the number of characters after the cursor, this shall not be considered
 108724 an error; all of the remaining characters shall be changed.

108725 **[count]_** Append a <space> after the current character position and then append the last
 108726 bigword in the previous input line after the <space>. Then enter insert mode after
 108727 the last character just appended. With a number *count*, append the *count*th bigword
 108728 in the previous line.

108729 **[count]x** Delete the character at the current cursor position and place the deleted characters
 108730 in the save buffer. If the cursor was positioned on the last character of the line, the
 108731 character shall be deleted and the cursor position shall be moved to the previous
 108732 character (the new last character). If the *count* is larger than the number of
 108733 characters after the cursor, this shall not be considered an error; all the characters
 108734 from the cursor to the end of the line shall be deleted.

108735 **[count]X** Delete the character before the current cursor position and place the deleted
 108736 characters in the save buffer. The character under the current cursor position shall
 108737 not change. If the cursor was positioned on the first character of the line, the
 108738 terminal shall be alerted, and the X command shall have no effect. If the line
 108739 contained a single character, the X command shall have no effect. If the line
 108740 contained no characters, the terminal shall be alerted and the cursor shall not be
 108741 moved. If the *count* is larger than the number of characters before the cursor, this
 108742 shall not be considered an error; all the characters from before the cursor to the
 108743 beginning of the line shall be deleted.

108744 **[count]dmotion**

108745 Delete the characters between the current cursor position and the character
 108746 position that would result from the motion command. A number *count* repeats the
 108747 motion command *count* times. If the motion command would move toward the
 108748 beginning of the command line, the character under the current cursor position
 108749 shall not be deleted. If the motion command is **d**, the entire current command line
 108750 shall be cleared. If the *count* is larger than the number of characters between the
 108751 current cursor position and the end of the command line toward which the motion
 108752 command would move the cursor, this shall not be considered an error; all of the
 108753 remaining characters in the aforementioned range shall be deleted. The deleted
 108754 characters shall be placed in the save buffer.

108755 **D** Delete all characters from the current cursor position to the end of the line. The
 108756 deleted characters shall be placed in the save buffer.

108757 **[count]ymotion**
 108758 Yank (that is, copy) the characters from the current cursor position to the position
 108759 resulting from the motion command into the save buffer. A number *count* shall be
 108760 applied to the motion command. If the motion command would move toward the
 108761 beginning of the command line, the character under the current cursor position
 108762 shall not be included in the set of yanked characters. If the motion command is **y**,
 108763 the entire current command line shall be yanked into the save buffer. The current
 108764 cursor position shall be unchanged. If the *count* is larger than the number of
 108765 characters between the current cursor position and the end of the command line
 108766 toward which the motion command would move the cursor, this shall not be
 108767 considered an error; all of the remaining characters in the aforementioned range
 108768 shall be yanked.

108769 **Y** Yank the characters from the current cursor position to the end of the line into the
 108770 save buffer. The current character position shall be unchanged.

108771 **[count]p** Put a copy of the current contents of the save buffer after the current cursor
 108772 position. The current cursor position shall be advanced to the last character put
 108773 from the save buffer. A *count* shall indicate how many copies of the save buffer
 108774 shall be put.

108775 **[count]P** Put a copy of the current contents of the save buffer before the current cursor
 108776 position. The current cursor position shall be moved to the last character put from
 108777 the save buffer. A *count* shall indicate how many copies of the save buffer shall be
 108778 put.

108779 **u** Undo the last command that changed the edit line. This operation shall not undo
 108780 the copy of any command line to the edit line.

108781 **U** Undo all changes made to the edit line. This operation shall not undo the copy of
 108782 any command line to the edit line.

108783 **[count]k**
 108784 **[count]-** Set the current command line to be the *count*th previous command line in the shell
 108785 command history. If *count* is not specified, it shall default to 1. The cursor shall be
 108786 positioned on the first character of the new command. If a **k** or **-** command would
 108787 retreat past the maximum number of commands in effect for this shell (affected by
 108788 the *HISTSIZE* environment variable), the terminal shall be alerted, and the
 108789 command shall have no effect.

108790 **[count]j**
 108791 **[count]+** Set the current command line to be the *count*th next command line in the shell
 108792 command history. If *count* is not specified, it shall default to 1. The cursor shall be
 108793 positioned on the first character of the new command. If a **j** or **+** command
 108794 advances past the edit line, the current command line shall be restored to the edit
 108795 line and the terminal shall be alerted.

108796 **[number]G** Set the current command line to be the oldest command line stored in the shell
 108797 command history. With a number *number*, set the current command line to be the
 108798 command line *number* in the history. If command line *number* does not exist, the
 108799 terminal shall be alerted and the command line shall not be changed.

108800 **/pattern<newline>**
 108801 Move backwards through the command history, searching for the specified
 108802 pattern, beginning with the previous command line. Patterns use the pattern
 108803 matching notation described in [Section 2.13](#) (on page 2382), except that the '^'

108804 character shall have special meaning when it appears as the first character of
 108805 *pattern*. In this case, the '^' is discarded and the characters after the '^' shall be
 108806 matched only at the beginning of a line. Commands in the command history shall
 108807 be treated as strings, not as filenames. If the pattern is not found, the current
 108808 command line shall be unchanged and the terminal shall be alerted. If it is found in
 108809 a previous line, the current command line shall be set to that line and the cursor
 108810 shall be set to the first character of the new command line.

108811 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If
 108812 there is no previous non-empty pattern, the terminal shall be alerted and the
 108813 current command line shall remain unchanged.

108814 **?pattern<newline>**

108815 Move forwards through the command history, searching for the specified pattern,
 108816 beginning with the next command line. Patterns use the pattern matching notation
 108817 described in [Section 2.13](#) (on page 2382), except that the '^' character shall have
 108818 special meaning when it appears as the first character of *pattern*. In this case, the
 108819 '^' is discarded and the characters after the '^' shall be matched only at the
 108820 beginning of a line. Commands in the command history shall be treated as strings,
 108821 not as filenames. If the pattern is not found, the current command line shall be
 108822 unchanged and the terminal shall be alerted. If it is found in a following line, the
 108823 current command line shall be set to that line and the cursor shall be set to the fist
 108824 character of the new command line.

108825 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If
 108826 there is no previous non-empty pattern, the terminal shall be alerted and the
 108827 current command line shall remain unchanged.

108828 **n** Repeat the most recent / or ? command. If there is no previous / or ?, the terminal
 108829 shall be alerted and the current command line shall remain unchanged.

108830 **N** Repeat the most recent / or ? command, reversing the direction of the search. If
 108831 there is no previous / or ?, the terminal shall be alerted and the current command
 108832 line shall remain unchanged.

108833 EXIT STATUS

108834 The following exit values shall be returned:

108835 0 The script to be executed consisted solely of zero or more blank lines or comments, or
 108836 both.

108837 1-125 A non-interactive shell detected an error other than *command_file* not found or
 108838 executable, including but not limited to syntax, redirection, or variable assignment
 108839 errors.

108840 126 A specified *command_file* could not be executed due to an [ENOEXEC] error (see [Section](#)
 108841 [2.9.1.1](#) (on page 2367), item 2).

108842 127 A specified *command_file* could not be found by a non-interactive shell.

108843 Otherwise, the shell shall return the exit status of the last command it invoked or attempted to
 108844 invoke (see also the *exit* utility in [Section 2.14](#), on page 2384).

108845 CONSEQUENCES OF ERRORS

108846 See [Section 2.8.1](#) (on page 2363).

108847 **APPLICATION USAGE**

108848 Standard input and standard error are the files that determine whether a shell is interactive
 108849 when `-i` is not specified. For example:

```
108850 sh > file
```

108851 and:

```
108852 sh 2> file
```

108853 create interactive and non-interactive shells, respectively. Although both accept terminal input,
 108854 the results of error conditions are different, as described in [Section 2.8.1](#) (on page 2363); in the
 108855 second example a redirection error encountered by a special built-in utility aborts the shell.

108856 A conforming application must protect its first operand, if it starts with a `<plus-sign>`, by
 108857 preceding it with the `--` argument that denotes the end of the options.

108858 Applications should note that the standard *PATH* to the shell cannot be assumed to be either
 108859 `/bin/sh` or `/usr/bin/sh`, and should be determined by interrogation of the *PATH* returned by
 108860 *getconf PATH*, ensuring that the returned pathname is an absolute pathname and not a shell
 108861 built-in.

108862 For example, to determine the location of the standard *sh* utility:

```
108863 command -v sh
```

108864 On some implementations this might return:

```
108865 /usr/xpg4/bin/sh
```

108866 Furthermore, on systems that support executable scripts (the `"#!"` construct), it is
 108867 recommended that applications using executable scripts install them using *getconf PATH* to
 108868 determine the shell pathname and update the `"#!"` script appropriately as it is being installed
 108869 (for example, with *sed*). For example:

```
108870 #
108871 # Installation time script to install correct POSIX shell pathname
108872 #
108873 # Get list of paths to check
108874 #
108875 Sifs=$IFS
108876 Sifs_set=${IFS+y}
108877 IFS=:
108878 set -- $(getconf PATH)
108879 if [ "$Sifs_set" = y ]
108880 then
108881     IFS=$Sifs
108882 else
108883     unset IFS
108884 fi
108885 #
108886 # Check each path for 'sh'
108887 #
108888 for i
108889 do
108890     if [ -x "${i}"/sh ]
108891     then
108892         Pshell=${i}/sh
```

```

108893         fi
108894     done
108895     #
108896     # This is the list of scripts to update. They should be of the
108897     # form '${name}.source' and will be transformed to '${name}'.
108898     # Each script should begin:
108899     #
108900     # #!INSTALLSHELLPATH
108901     #
108902     scripts="a b c"
108903     #
108904     # Transform each script
108905     #
108906     for i in ${scripts}
108907     do
108908         sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
108909     done

```

108910 EXAMPLES

- 108911 1. Execute a shell command from a string:


```
108912     sh -c "cat myfile"
```
- 108913 2. Execute a shell script from a file in the current directory:


```
108914     sh my_shell_cmds
```

108915 RATIONALE

108916 The *sh* utility and the *set* special built-in utility share a common set of options.

108917 The name *IFS* was originally an abbreviation of “Input Field Separators”; however, this name is
 108918 misleading as the *IFS* characters are actually used as field terminators. One justification for
 108919 ignoring the contents of *IFS* upon entry to the script, beyond security considerations, is to assist
 108920 possible future shell compilers. Allowing *IFS* to be imported from the environment prevents
 108921 many optimizations that might otherwise be performed via dataflow analysis of the script itself.

108922 The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been
 108923 invoked, probably by a C-language program, with standard input that has been opened using
 108924 the `O_NONBLOCK` flag; see *open()* in the System Interfaces volume of POSIX.1-2017. If the shell
 108925 did not reset this flag, it would immediately terminate because no input data would be available
 108926 yet and that would be considered the same as end-of-file.

108927 The options associated with a *restricted shell* (command name *rsh* and the `-r` option) were
 108928 excluded because the standard developers considered that the implied level of security could
 108929 not be achieved and they did not want to raise false expectations.

108930 On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the
 108931 name `-i`. When it is called by a sequence such as:

```
108932 sh -
```

108933 or by:

```
108934 #! usr/bin/sh -
```

108935 the historical systems have assumed that no option letters follow. Thus, this volume of
 108936 POSIX.1-2017 allows the single `<hyphen-minus>` to mark the end of the options, in addition to
 108937 the use of the regular `"--"` argument, because it was considered that the older practice was so

108938 pervasive. An alternative approach is taken by the KornShell, where real and effective
108939 user/group IDs must match for an interactive shell; this behavior is specifically allowed by this
108940 volume of POSIX.1-2017.

108941 **Note:** There are other problems with set-user-ID scripts that the two approaches described here do not
108942 resolve.

108943 The initialization process for the history file can be dependent on the system start-up files, in
108944 that they may contain commands that effectively preempt the user's settings of *HISTFILE* and
108945 *HISTSIZE*. For example, function definition commands are recorded in the history file, unless
108946 the *set -o nolog* option is set. If the system administrator includes function definitions in some
108947 system start-up file called before the *ENV* file, the history file is initialized before the user gets a
108948 chance to influence its characteristics. In some historical shells, the history file is initialized just
108949 after the *ENV* file has been processed. Therefore, it is implementation-defined whether changes
108950 made to *HISTFILE* after the history file has been initialized are effective.

108951 The default messages for the various *MAIL*-related messages are unspecified because they vary
108952 across implementations. Typical messages are:

108953 "you have mail\n"

108954 or:

108955 "you have new mail\n"

108956 It is important that the descriptions of command line editing refer to the same shell as that in
108957 POSIX.1-2017 so that interactive users can also be application programmers without having to
108958 deal with programmatic differences in their two environments. It is also essential that the utility
108959 name *sh* be specified because this explicit utility name is too firmly rooted in historical practice
108960 of application programs for it to change.

108961 Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on
108962 terminals that do not support command line editing. However, it is not historical practice for the
108963 shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in
108964 all cases. Implementations are encouraged to supply diagnostics in this case whenever possible,
108965 rather than leaving the user in a state where editing commands work incorrectly.

108966 In early proposals, the KornShell-derived *emacs* mode of command line editing was included,
108967 even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant
108968 that the full *emacs* editor not be standardized because they were concerned that an attempt to
108969 standardize this very powerful environment would encourage vendors to ship strictly
108970 conforming versions lacking the extensibility required by the community. The author of the
108971 original *emacs* program also expressed his desire to omit the program. Furthermore, there were a
108972 number of historical systems that did not include *emacs*, or included it without supporting it, but
108973 there were very few that did not include and support *vi*. The shell *emacs* command line editing
108974 mode was finally omitted because it became apparent that the KornShell version and the editor
108975 being distributed with the GNU system had diverged in some respects. The author of *emacs*
108976 requested that the POSIX *emacs* mode either be deleted or have a significant number of
108977 unspecified conditions. Although the KornShell author agreed to consider changes to bring the
108978 shell into alignment, the standard developers decided to defer specification at that time. At the
108979 time, it was assumed that convergence on an acceptable definition would occur for a subsequent
108980 draft, but that has not happened, and there appears to be no impetus to do so. In any case,
108981 implementations are free to offer additional command line editing modes based on the exact
108982 models of editors their users are most comfortable with.

108983 Early proposals had the following list entry in [vi Line Editing Insert Mode](#) (on page 3231):

108984 \ If followed by the *erase* or *kill* character, that character shall be inserted into the input line.
 108985 Otherwise, the <backslash> itself shall be inserted into the input line.

108986 However, this is not actually a feature of *sh* command line editing insert mode, but one of some
 108987 historical terminal line drivers. Some conforming implementations continue to do this when the
 108988 *stty ixtext* flag is set.

108989 In interactive shells, SIGTERM is ignored so that `kill 0` does not kill the shell, and SIGINT is
 108990 caught so that *wait* is interruptible. If the shell does not ignore SIGTTIN, SIGTTOU, and
 108991 SIGTSTP signals when it is interactive and the `-m` option is not in effect, these signals suspend
 108992 the shell if it is not a session leader. If it is a session leader, the signals are discarded if they
 108993 would stop the process, as required by XSH Section 2.4.3 (on page 490) for orphaned process
 108994 groups.

108995 FUTURE DIRECTIONS

108996 None.

108997 SEE ALSO

108998 Section 2.9.1.1 (on page 2367), Chapter 2 (on page 2345), *cd*, *echo*, *exit*, *fc*, *pwd*, *invalid*, *set*, *stty*,
 108999 *test*, *trap*, *umask*, *vi*

109000 XBD Chapter 8 (on page 173), Section 12.2 (on page 216)

109001 XSH *dup()*, *exec*, *exit()*, *fork()*, *open()*, *pipe()*, *signal()*, *system()*, *ulimit()*, *umask()*, *wait()*

109002 CHANGE HISTORY

109003 First released in Issue 2.

109004 Issue 5

109005 The FUTURE DIRECTIONS section is added.

109006 Text is added to the DESCRIPTION for the Large File Summit proposal.

109007 Issue 6

109008 The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

109009 The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

109010 The following new requirements on POSIX implementations derive from alignment with the
 109011 Single UNIX Specification:

109012 The option letters derived from the *set* special built-in are also accepted with a leading
 109013 <plus-sign> ('+').

109014 Large file extensions are added:

109015 `stat` filename expansion does not fail due to the size of a file.

109016 `head` input and output redirections have an implementation-defined offset maximum
 109017 that is established in the open file description.

109018 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to
 109019 “directory referred to by the *HOME* environment variable”.

109020 Descriptions for the *ENV* and *PWD* environment variables are included to align with the
 109021 IEEE P1003.2b draft standard.

109022 The normative text is reworded to avoid use of the term “must” for application requirements.

109023 **Issue 7**

- 109024 Austin Group Interpretation 1003.1-2001 #098 is applied, changing the definition of *IFS*.
- 109025 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 109026 Changes to the *pwd* utility and *PWD* environment variable have been made to match the
109027 changes to the *getcwd()* function made for Austin Group Interpretation 1003.1-2001 #140.
- 109028 Minor editorial changes are made to the User Portability Utilities option shading. No normative
109029 changes are implied.
- 109030 Minor changes are made to the install script example in the APPLICATION USAGE section.
- 109031 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0137 [152], XCU/TC1-2008/0138
109032 [347], XCU/TC1-2008/0139 [347], XCU/TC1-2008/0140 [347], XCU/TC1-2008/0141 [299], and
109033 XCU/TC1-2008/0142 [347] are applied.
- 109034 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0175 [584], XCU/TC2-2008/0176
109035 [584], XCU/TC2-2008/0177 [718], XCU/TC2-2008/0178 [884], XCU/TC2-2008/0179 [809],
109036 XCU/TC2-2008/0180 [884], and XCU/TC2-2008/0181 [584] are applied.

109037 **NAME**

109038 sleep ‡suspend execution for an interval

109039 **SYNOPSIS**

109040 sleep *time*

109041 **DESCRIPTION**

109042 The *sleep* utility shall suspend execution for at least the integral number of seconds specified by
109043 the *time* operand.

109044 **OPTIONS**

109045 None.

109046 **OPERANDS**

109047 The following operand shall be supported:

109048 *time* A non-negative decimal integer specifying the number of seconds for which to
109049 suspend execution.

109050 **STDIN**

109051 Not used.

109052 **INPUT FILES**

109053 None.

109054 **ENVIRONMENT VARIABLES**

109055 The following environment variables shall affect the execution of *sleep*:

109056 *LANG* Provide a default value for the internationalization variables that are unset or null.
109057 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
109058 variables used to determine the values of locale categories.)

109059 *LC_ALL* If set to a non-empty string value, override the values of all the other
109060 internationalization variables.

109061 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
109062 characters (for example, single-byte as opposed to multi-byte characters in
109063 arguments).

109064 *LC_MESSAGES*

109065 Determine the locale that should be used to affect the format and contents of
109066 diagnostic messages written to standard error.

109067 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

109068 **ASYNCHRONOUS EVENTS**

109069 If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

- 109070 1. Terminate normally with a zero exit status.
- 109071 2. Effectively ignore the signal.
- 109072 3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS
109073 section of [Section 1.4](#) (on page 2336). This could include terminating with a non-zero exit
109074 status.

109075 The *sleep* utility shall take the standard action for all other signals.

109076 **STDOUT**

109077 Not used.

109078 **STDERR**

109079 The standard error shall be used only for diagnostic messages.

109080 **OUTPUT FILES**

109081 None.

109082 **EXTENDED DESCRIPTION**

109083 None.

109084 **EXIT STATUS**

109085 The following exit values shall be returned:

109086 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal
109087 was received. See the ASYNCHRONOUS EVENTS section.

109088 >0 An error occurred.

109089 **CONSEQUENCES OF ERRORS**

109090 Default.

109091 **APPLICATION USAGE**

109092 None.

109093 **EXAMPLES**109094 The *sleep* utility can be used to execute a command after a certain amount of time, as in:109095 `(sleep 105; command) &`

109096 or to execute a command every so often, as in:

109097 `while true`
109098 `do`
109099 `command`
109100 `sleep 37`
109101 `done`109102 **RATIONALE**109103 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because
109104 most implementations of this utility rely on the arrival of that signal to notify them that the
109105 requested finishing time has been successfully attained. Such implementations thus do not
109106 distinguish this situation from the successful completion case. Other implementations are
109107 allowed to catch the signal and go back to sleep until the requested time expires or to provide
109108 the normal signal termination procedures.109109 As with all other utilities that take integral operands and do not specify subranges of allowed
109110 values, *sleep* is required by this volume of POSIX.1-2017 to deal with *time* requests of up to
109111 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls to
109112 the delay mechanism of the underlying operating system if its argument range is less than this.109113 **FUTURE DIRECTIONS**

109114 None.

109115 **SEE ALSO**109116 *wait*

109117 XBD Chapter 8 (on page 173)

109118 XSH *alarm()*, *sleep()*



109119 **CHANGE HISTORY**

109120 First released in Issue 2.



109121 **NAME**109122 `sort` — sort, merge, or sequence check text files109123 **SYNOPSIS**109124 `sort [-m] [-o output] [-bdfinru] [-t char] [-k keydef]... [file...]`109125 `sort [-c|-C] [-bdfinru] [-t char] [-k keydef] [file]`109126 **DESCRIPTION**109127 The `sort` utility shall perform one of the following functions:

- 109128 1. Sort lines of all the named files together and write the result to the specified output.
- 109129 2. Merge lines of all the named (presorted) files together and write the result to the specified
109130 output.
- 109131 3. Check that a single input file is correctly presorted.

109132 Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no
109133 sort keys are specified, the entire line up to, but not including, the terminating <newline>), and
109134 shall be performed using the collating sequence of the current locale. If this collating sequence
109135 does not have a total ordering of all characters (see XBD Section 7.3.2, on page 147), any lines of
109136 input that collate equally should be further compared byte-by-byte using the collating sequence
109137 for the POSIX locale.

109138 **OPTIONS**

109139 The `sort` utility shall conform to XBD Section 12.2 (on page 216), except for Guideline 9, and the
109140 `-k keydef` option should follow the `-b`, `-d`, `-f`, `-i`, `-n`, and `-r` options. In addition, '+' may be
109141 recognized as an option delimiter as well as '-'.

109142 The following options shall be supported:

- 109143 `-c` Check that the single input file is ordered as specified by the arguments and the
109144 collating sequence of the current locale. Output shall not be sent to standard
109145 output. The exit code shall indicate whether or not disorder was detected or an
109146 error occurred. If disorder (or, with `-u`, a duplicate key) is detected, a warning
109147 message shall be sent to standard error indicating where the disorder or duplicate
109148 key was found.
- 109149 `-C` Same as `-c`, except that a warning message shall not be sent to standard error if
109150 disorder or, with `-u`, a duplicate key is detected.
- 109151 `-m` Merge only; the input file shall be assumed to be already sorted.
- 109152 `-o output` Specify the name of an output file to be used instead of the standard output. This
109153 file can be the same as one of the input *files*.
- 109154 `-u` Unique: suppress all but one in each set of lines having equal keys. If used with
109155 the `-c` option, check that there are no lines with duplicate keys, in addition to
109156 checking that the input file is sorted.

109157 The following options shall override the default ordering rules. When ordering options appear
109158 independent of any key field specifications, the requested field ordering rules shall be applied
109159 globally to all sort keys. When attached to a specific key (see `-k`), the specified ordering options
109160 shall override all global ordering options for that key.

- 109161 `-d` Specify that only <blank> characters and alphanumeric characters, according to
109162 the current setting of `LC_CTYPE`, shall be significant in comparisons. The behavior
109163 is undefined for a sort key to which `-i` or `-n` also applies.

- 109164 **-f** Consider all lowercase characters that have uppercase equivalents, according to
109165 the current setting of *LC_CTYPE*, to be the uppercase equivalent for the purposes
109166 of comparison.
- 109167 **-i** Ignore all characters that are non-printable, according to the current setting of
109168 *LC_CTYPE*. The behavior is undefined for a sort key for which **-n** also applies.
- 109169 **-n** Restrict the sort key to an initial numeric string, consisting of optional <blank>
109170 characters, optional <hyphen-minus> character, and zero or more digits with an
109171 optional radix character and thousands separators (as defined in the current
109172 locale), which shall be sorted by arithmetic value. An empty digit string shall be
109173 treated as zero. Leading zeros and signs on zeros shall not affect ordering.
- 109174 **-r** Reverse the sense of comparisons.
- 109175 The treatment of field separators can be altered using the options:
- 109176 **-b** Ignore leading <blank> characters when determining the starting and ending
109177 positions of a restricted sort key. If the **-b** option is specified before the first **-k**
109178 option, it shall be applied to all **-k** options. Otherwise, the **-b** option can be
109179 attached independently to each **-k** *field_start* or *field_end* option-argument (see
109180 below).
- 109181 **-t char** Use *char* as the field separator character; *char* shall not be considered to be part of a
109182 field (although it can be included in a sort key). Each occurrence of *char* shall be
109183 significant (for example, <*char*><*char*> delimits an empty field). If **-t** is not
109184 specified, <blank> characters shall be used as default field separators; each
109185 maximal non-empty sequence of <blank> characters that follows a non-<blank>
109186 shall be a field separator.
- 109187 Sort keys can be specified using the options:
- 109188 **-k keydef** The *keydef* argument is a restricted sort key field definition. The format of this
109189 definition is:
- 109190 *field_start*[*type*][,*field_end*[*type*]]
- 109191 where *field_start* and *field_end* define a key field restricted to a portion of the line
109192 (see the EXTENDED DESCRIPTION section), and *type* is one or more modifiers
109193 from the list of characters 'b', 'd', 'f', 'i', 'n', 'r'. The 'b' modifier shall
109194 behave like the **-b** option, but shall apply only to the *field_start* or *field_end* to
109195 which it is attached. The other modifiers shall behave like the corresponding
109196 options, but shall apply only to the key field to which they are attached; they shall
109197 have this effect if specified with *field_start*, *field_end*, or both. If any modifier is
109198 attached to a *field_start* or to a *field_end*, no option shall apply to either.
109199 Implementations shall support at least nine occurrences of the **-k** option, which
109200 shall be significant in command line order. If no **-k** option is specified, a default
109201 sort key of the entire line shall be used.
- 109202 When there are multiple key fields, later keys shall be compared only after all
109203 earlier keys compare equal. Except when the **-u** option is specified, lines that
109204 otherwise compare equal shall be ordered as if none of the options **-d**, **-f**, **-i**, **-n**, or
109205 **-k** were present (but with **-r** still in effect, if it was specified) and with all bytes in
109206 the lines significant to the comparison. The order in which lines that still compare
109207 equal are written is unspecified.

109208 **OPERANDS**

109209 The following operand shall be supported:

109210 *file* A pathname of a file to be sorted, merged, or checked. If no *file* operands are
 109211 specified, or if a *file* operand is '-', the standard input shall be used. If *sort*
 109212 encounters an error when opening or reading a *file* operand, it may exit without
 109213 writing any output to standard output or processing later operands.

109214 **STDIN**

109215 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
 109216 See the INPUT FILES section.

109217 **INPUT FILES**

109218 The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a
 109219 file ending with an incomplete last line.

109220 **ENVIRONMENT VARIABLES**

109221 The following environment variables shall affect the execution of *sort*:

109222 *LANG* Provide a default value for the internationalization variables that are unset or null.
 109223 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 109224 variables used to determine the values of locale categories.)

109225 *LC_ALL* If set to a non-empty string value, override the values of all the other
 109226 internationalization variables.

109227 *LC_COLLATE*

109228 Determine the locale for ordering rules.

109229 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 109230 characters (for example, single-byte as opposed to multi-byte characters in
 109231 arguments and input files) and the behavior of character classification for the **-b**,
 109232 **-d**, **-f**, **-i**, and **-n** options.

109233 *LC_MESSAGES*

109234 Determine the locale that should be used to affect the format and contents of
 109235 diagnostic messages written to standard error.

109236 *LC_NUMERIC*

109237 Determine the locale for the definition of the radix character and thousands
 109238 separator for the **-n** option.

109239 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

109240 **ASYNCHRONOUS EVENTS**

109241 Default.

109242 **STDOUT**

109243 Unless the **-o** or **-c** options are in effect, the standard output shall contain the sorted input.

109244 **STDERR**

109245 The standard error shall be used for diagnostic messages. When **-c** is specified, if disorder is
 109246 detected (or if **-u** is also specified and a duplicate key is detected), a message shall be written to
 109247 the standard error which identifies the input line at which disorder (or a duplicate key) was
 109248 detected. A warning message about correcting an incomplete last line of an input file may be
 109249 generated, but need not affect the final exit status.

109250 **OUTPUT FILES**

109251 If the `-o` option is in effect, the sorted input shall be written to the file *output*.

109252 **EXTENDED DESCRIPTION**

109253 The notation:

109254 `-k field_start[type][,field_end[type]]`

109255 shall define a key field that begins at *field_start* and ends at *field_end* inclusive, unless *field_start*
109256 falls beyond the end of the line or after *field_end*, in which case the key field is empty. A missing
109257 *field_end* shall mean the last character of the line.

109258 A field comprises a maximal sequence of non-separating characters and, in the absence of option
109259 `-t`, any preceding field separator.

109260 The *field_start* portion of the *keydef* option-argument shall have the form:

109261 `field_number[.first_character]`

109262 Fields and characters within fields shall be numbered starting with 1. The *field_number* and
109263 *first_character* pieces, interpreted as positive decimal integers, shall specify the first character to
109264 be used as part of a sort key. If *first_character* is omitted, it shall refer to the first character of the
109265 field.

109266 The *field_end* portion of the *keydef* option-argument shall have the form:

109267 `field_number[.last_character]`

109268 The *field_number* shall be as described above for *field_start*. The *last_character* piece, interpreted
109269 as a non-negative decimal integer, shall specify the last character to be used as part of the sort
109270 key. If *last_character* evaluates to zero or *last_character* is omitted, it shall refer to the last
109271 character of the field specified by *field_number*.

109272 If the `-b` option or `b` type modifier is in effect, characters within a field shall be counted from the
109273 first non-`<blank>` in the field. (This shall apply separately to *first_character* and *last_character*.)

109274 **EXIT STATUS**

109275 The following exit values shall be returned:

109276 0 All input files were output successfully, or `-c` was specified and the input file was correctly
109277 sorted.

109278 1 Under the `-c` option, the file was not ordered as specified, or if the `-c` and `-u` options were
109279 both specified, two input lines were found with equal keys.

109280 `>1` An error occurred.

109281 **CONSEQUENCES OF ERRORS**

109282 The default requirements shall apply, except that if *sort* encounters an error when opening or
109283 reading a *file* operand, it may exit without writing any output to standard output or processing
109284 later operands.

109285 **APPLICATION USAGE**

109286 The default value for `-t`, `<blank>`, has different properties from, for example, `-t"<space>"`. If a
 109287 line contains:

109288 `<space><space>foo`

109289 the following treatment would occur with default separation as opposed to specifically selecting
 109290 a `<space>`:

Field	Default	<code>-t "<space>"</code>
1	<code><space><space>foo</code>	<i>empty</i>
2	<i>empty</i>	<i>empty</i>
3	<i>empty</i>	foo

109295 The leading field separator itself is included in a field when `-t` is not used. For example, this
 109296 command returns an exit status of zero, meaning the input was already sorted:

```
109297 sort -c -k 2 <<eof
109298 y<tab>b
109299 x<space>a
109300 eof
```

109301 (assuming that a `<tab>` precedes the `<space>` in the current collating sequence). The field
 109302 separator is not included in a field when it is explicitly set via `-t`. This is historical practice and
 109303 allows usage such as:

```
109304 sort -t "|" -k 2n <<eof
109305 Atlanta|425022|Georgia
109306 Birmingham|284413|Alabama
109307 Columbia|100385|South Carolina
109308 eof
```

109309 where the second field can be correctly sorted numerically without regard to the non-numeric
 109310 field separator.

109311 The wording in the OPTIONS section clarifies that the `-b`, `-d`, `-f`, `-i`, `-n`, and `-r` options have to
 109312 come before the first sort key specified if they are intended to apply to all specified keys. The
 109313 way it is described in this volume of POSIX.1-2017 matches historical practice, not historical
 109314 documentation. The results are unspecified if these options are specified after a `-k` option.

109315 The `-f` option might not work as expected in locales where there is not a one-to-one mapping
 109316 between an uppercase and a lowercase letter.

109317 When using `sort` to process pathnames, it is recommended that `LC_ALL`, or at least `LC_CTYPE`
 109318 and `LC_COLLATE`, are set to `POSIX` or `C` in the environment, since pathnames can contain byte
 109319 sequences that do not form valid characters in some locales, in which case the utility's behavior
 109320 would be undefined. In the `POSIX` locale each byte is a valid single-byte character, and therefore
 109321 this problem is avoided.

109322 If the collating sequence of the current locale does not have a total ordering of all characters, this
 109323 can affect the behavior of `sort` in the following ways:

109324 As `sort -u` suppresses lines with duplicate keys, it suppresses lines that collate equally but
 109325 are not identical.

109326 The output of `sort` (without `-u`) can contain identical lines that are not adjacent, if it does
 109327 not implement the recommended further byte-by-byte comparison of lines that collate
 109328 equally. This affects the use of `sort` with `comm` and `uniq`; see the APPLICATION USAGE for

109329 those utilities.

109330 EXAMPLES

109331 1. The following command sorts the contents of **infile** with the second field as the sort key:

```
109332 sort -k 2,2 infile
```

109333 2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**,
109334 placing the output in **outfile** and using the second character of the second field as the sort
109335 key (assuming that the first character of the second field is the field separator):

```
109336 sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

109337 3. The following command sorts the contents of **infile1** and **infile2** using the second
109338 non-<blank> of the second field as the sort key:

```
109339 sort -k 2.2b,2.2b infile1 infile2
```

109340 4. The following command prints the System V password file (user database) sorted by the
109341 numeric user ID (the third <colon>-separated field):

```
109342 sort -t : -k 3,3n /etc/passwd
```

109343 5. The following command prints the lines of the already sorted file **infile**, suppressing all
109344 but one occurrence of lines having the same third field:

```
109345 sort -um -k 3.1,3.0 infile
```

109346 RATIONALE

109347 Examples in some historical documentation state that options **-um** with one input file keep the
109348 first in each set of lines with equal keys. This behavior was deemed to be an implementation
109349 artifact and was not standardized.

109350 The **-z** option was omitted; it is not standard practice on most systems and is inconsistent with
109351 using *sort* to sort several files individually and then merge them together. The text concerning **-z**
109352 in historical documentation appeared to require implementations to determine the proper buffer
109353 length during the sort phase of operation, but not during the merge.

109354 The **-y** option was omitted because of non-portability. The **-M** option, present in System V, was
109355 omitted because of non-portability in international usage.

109356 An undocumented **-T** option exists in some implementations. It is used to specify a directory for
109357 intermediate files. Implementations are encouraged to support the use of the *TMPDIR*
109358 environment variable instead of adding an option to support this functionality.

109359 The **-k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is
109360 not consistent with other utility conventions. Second, it did not meet syntax guideline
109361 requirements.

109362 Historical documentation indicates that “setting **-n** implies **-b**”. The description of **-n** already
109363 states that optional leading <blank>s are tolerated in doing the comparison. If **-b** is enabled,
109364 rather than implied, by **-n**, this has unusual side-effects. When a character offset is used in a
109365 column of numbers (for example, to sort modulo 100), that offset is measured relative to the
109366 most significant digit, not to the column. Based upon a recommendation from the author of the
109367 original *sort* utility, the **-b** implication has been omitted from this volume of POSIX.1-2017, and
109368 an application wishing to achieve the previously mentioned side-effects has to code the **-b** flag
109369 explicitly.

109370 Earlier versions of this standard allowed the **-o** option to appear after operands. Historical
109371 practice allowed all options to be interspersed with operands. This version of the standard

109372 allows implementations to accept options after operands but conforming applications should
109373 not use this form.

109374 Earlier versions of this standard also allowed the *-number* and *+number* options. These options
109375 are no longer specified by POSIX.1-2017 but may be present in some implementations.

109376 Historical implementations produced a message on standard error when *-c* was specified and
109377 disorder was detected, and when *-c* and *-u* were specified and a duplicate key was detected. An
109378 earlier version of this standard contained wording that did not make it clear that this message
109379 was allowed and some implementations removed this message to be sure that they conformed
109380 to the standard's requirements. Confronted with this difference in behavior, interactive users
109381 that wanted to be sure that they got visual feedback instead of just exit code 1 could have used a
109382 command like:

```
109383 sort -c file || echo disorder
```

109384 whether or not the *sort* utility provided a message in this case. But, it was not easy for a user to
109385 find where the disorder or duplicate key occurred on implementations that do not produce a
109386 message, especially when some parts of the input line were not part of the key and when one or
109387 more of the *-b*, *-d*, *-f*, *-i*, *-n*, or *-r* options or *keydef* type modifiers were in use. POSIX.1-2017
109388 requires a message to be produced in this case. POSIX.1-2017 also contains the *-C* option giving
109389 users the ability to choose either behavior.

109390 When a disorder or duplicate is found when the *-c* option is specified, some implementations
109391 print a message containing the first line that is out of order or contains a duplicate key; others
109392 print a message specifying the line number of the offending line. This standard allows either
109393 type of message.

109394 Implementations are encouraged to perform the recommended further byte-by-byte comparison
109395 of lines that collate equally, even though this may affect efficiency. The impact on efficiency can
109396 be mitigated by only performing the additional comparison if the current locale's collating
109397 sequence does not have a total ordering of all characters (if the implementation provides a way
109398 to query this) or by only performing the additional comparison if the locale name associated
109399 with the LC_COLLATE category has an '@' modifier in the name (since locales without an '@'
109400 modifier should have a total ordering of all characters — see XBD [Section 7.3.2](#), on page 147).
109401 Note that if the implementation provides a *stable sort* option as an extension (usually *-s*), the
109402 additional comparison should not be performed when this option has been specified.

109403 FUTURE DIRECTIONS

109404 A future version of this standard may require that if the collating sequence of the current locale
109405 does not have a total ordering of all characters, any lines of input that collate equally when
109406 comparing them as whole lines are further compared byte-by-byte using the collating sequence
109407 for the POSIX locale.

109408 SEE ALSO

109409 *comm*, *join*, *uniq*

109410 XBD [Section 7.3.2](#) (on page 147), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

109411 XSH *toupper()*

109412 CHANGE HISTORY

109413 First released in Issue 2.

109414 **Issue 6**

109415 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

109416 IEEE PASC Interpretation 1003.2 #168 is applied.

109417 **Issue 7**

109418 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply and noting that '+' may be recognized as an option delimiter.

109420 Austin Group Interpretation 1003.1-2001 #120 is applied, clarifying the use of the -c option and introducing the -C option.

109422 XCU-ERN-81 is applied, modifying the description of the -i option.

109423 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

109424 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0182 [963], XCU/TC2-2008/0183 [584], XCU/TC2-2008/0184 [510], XCU/TC2-2008/0185 [962], XCU/TC2-2008/0186 [663], and XCU/TC2-2008/0187 [963] are applied.

109427 **NAME**109428 `split` ‡split a file into pieces109429 **SYNOPSIS**109430 `split [-l line_count] [-a suffix_length] [file [name]]`109431 `split -b n[k|m] [-a suffix_length] [file [name]]`109432 **DESCRIPTION**

109433 The *split* utility shall read an input file and write zero or more output files. The default size of
 109434 each output file shall be 1 000 lines. The size of the output files can be modified by specification
 109435 of the `-b` or `-l` options. Each output file shall be created with a unique suffix. The suffix shall
 109436 consist of exactly *suffix_length* lowercase letters from the POSIX locale. The letters of the suffix
 109437 shall be used as if they were a base-26 digit system, with the first suffix to be created consisting
 109438 of all 'a' characters, the second with a 'b' replacing the last 'a', and so on, until a name of all
 109439 'z' characters is created. By default, the names of the output files shall be 'x', followed by a
 109440 two-character suffix from the character set as described above, starting with "aa", "ab", "ac",
 109441 and so on, and continuing until the suffix "zz", for a maximum of 676 files.

109442 If the number of files required exceeds the maximum allowed by the suffix length provided,
 109443 such that the last allowable file would be larger than the requested size, the *split* utility shall fail
 109444 after creating the last file with a valid suffix; *split* shall not delete the files it created with valid
 109445 suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the
 109446 input file, and may be smaller than the requested size. If the input is an empty file, no output file
 109447 shall be created and this shall not be considered to be an error.

109448 **OPTIONS**109449 The *split* utility shall conform to XBD [Section 12.2](#) (on page 216).

109450 The following options shall be supported:

109451 `-a suffix_length`

109452 Use *suffix_length* letters to form the suffix portion of the filenames of the split file. If
 109453 `-a` is not specified, the default suffix length shall be two. If the sum of the *name*
 109454 operand and the *suffix_length* option-argument would create a filename exceeding
 109455 {NAME_MAX} bytes, an error shall result; *split* shall exit with a diagnostic message
 109456 and no files shall be created.

109457 `-b n` Split a file into pieces *n* bytes in size.109458 `-b nk` Split a file into pieces *n**1 024 bytes in size.109459 `-b nm` Split a file into pieces *n**1 048 576 bytes in size.

109460 `-l line_count` Specify the number of lines in each resulting file piece. The *line_count* argument is
 109461 an unsigned decimal integer. The default is 1 000. If the input does not end with a
 109462 <newline>, the partial line shall be included in the last output file.

109463 **OPERANDS**

109464 The following operands shall be supported:

109465 *file* The pathname of the ordinary file to be split. If no input file is given or *file* is '-',
 109466 the standard input shall be used.

109467 *name* The prefix to be used for each of the files resulting from the split operation. If no
 109468 *name* argument is given, 'x' shall be used as the prefix of the output files. The
 109469 combined length of the basename of *prefix* and *suffix_length* cannot exceed
 109470 {NAME_MAX} bytes. See the OPTIONS section.

109471 **STDIN**

109472 See the INPUT FILES section.

109473 **INPUT FILES**

109474 Any file can be used as input.

109475 **ENVIRONMENT VARIABLES**

109476 The following environment variables shall affect the execution of *split*:

109477 **LANG** Provide a default value for the internationalization variables that are unset or null.
109478 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
109479 variables used to determine the values of locale categories.)

109480 **LC_ALL** If set to a non-empty string value, override the values of all the other
109481 internationalization variables.

109482 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
109483 characters (for example, single-byte as opposed to multi-byte characters in
109484 arguments and input files).

109485 **LC_MESSAGES**

109486 Determine the locale that should be used to affect the format and contents of
109487 diagnostic messages written to standard error.

109488 **XSI NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

109489 **ASYNCHRONOUS EVENTS**

109490 Default.

109491 **STDOUT**

109492 Not used.

109493 **STDERR**

109494 The standard error shall be used only for diagnostic messages.

109495 **OUTPUT FILES**

109496 The output files contain portions of the original input file; otherwise, unchanged.

109497 **EXTENDED DESCRIPTION**

109498 None.

109499 **EXIT STATUS**

109500 The following exit values shall be returned:

109501 0 Successful completion.

109502 >0 An error occurred.

109503 **CONSEQUENCES OF ERRORS**

109504 Default.

109505 **APPLICATION USAGE**

109506 None.

109507 **EXAMPLES**109508 In the following examples **foo** is a text file that contains 5 000 lines.109509 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:109510

```
split foo
```

109511 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,
109512 **xaab**, **xaac**, **xaad**, and **xaae**:109513

```
split -a 3 foo
```

109514 3. Create three files with four-letter suffixes and a supplied prefix, **bar_aaaa**, **bar_aaab**, and
109515 **bar_aaac**:109516

```
split -a 4 -l 2000 foo bar_
```

109517 4. Create as many files as are necessary to contain at most 20*1 024 bytes, each with the
109518 default prefix of **x** and a five-letter suffix:109519

```
split -a 5 -b 20k foo
```

109520 **RATIONALE**109521 The **-b** option was added to provide a mechanism for splitting files other than by lines. While
109522 most uses of the **-b** option are for transmitting files over networks, some believed it would have
109523 additional uses.109524 The **-a** option was added to overcome the limitation of being able to create only 676 files.109525 Consideration was given to deleting this utility, using the rationale that the functionality
109526 provided by this utility is available via the *csplit* utility (see *csplit*). Upon reconsideration of the
109527 purpose of the User Portability Utilities option, it was decided to retain both this utility and the
109528 *csplit* utility because users use both utilities and have historical expectations of their behavior.
109529 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical
109530 *csplit*.109531 The text “*split* shall not delete the files it created with valid suffixes” would normally be
109532 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the
109533 historical behavior of *split* is made explicit to avoid misinterpretation.109534 Earlier versions of this standard allowed a *-line_count* option. This form is no longer specified by
109535 POSIX.1-2017 but may be present in some implementations.109536 **FUTURE DIRECTIONS**

109537 None.

109538 **SEE ALSO**109539 *csplit*109540 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)109541 **CHANGE HISTORY**

109542 First released in Issue 2.

109543 **Issue 6**

109544 This utility is marked as part of the User Portability Utilities option.

109545 The APPLICATION USAGE section is added.

- 109546 The obsolescent SYNOPSIS is removed.
- 109547 **Issue 7**
- 109548 Austin Group Interpretation 1003.1-2001 #027 is applied.
- 109549 The *split* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.
- 109550
- 109551 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 109552 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0188 [731] is applied.

109553 **NAME**

109554 strings ‡find printable strings in files

109555 **SYNOPSIS**109556 strings [-a] [-t *format*] [-n *number*] [*file...*]109557 **DESCRIPTION**

109558 The *strings* utility shall look for printable strings in regular files and shall write those strings to
 109559 standard output. A printable string is any sequence of four (by default) or more printable
 109560 characters terminated by a <newline> or NUL character. Additional implementation-defined
 109561 strings may be written; see *localedef*.

109562 If the first argument is '-', the results are unspecified.

109563 **OPTIONS**

109564 The *strings* utility shall conform to XBD [Section 12.2](#) (on page 216), except for the unspecified
 109565 usage of '-'.

109566 The following options shall be supported:

109567 **-a** Scan files in their entirety. If **-a** is not specified, it is implementation-defined what
 109568 portion of each file is scanned for strings.

109569 **-n *number*** Specify the minimum string length, where the *number* argument is a positive
 109570 decimal integer. The default shall be 4.

109571 **-t *format*** Write each string preceded by its byte offset from the start of the file. The format
 109572 shall be dependent on the single character used as the *format* option-argument:

109573 d The offset shall be written in decimal.

109574 o The offset shall be written in octal.

109575 x The offset shall be written in hexadecimal.

109576 **OPERANDS**

109577 The following operand shall be supported:

109578 *file* A pathname of a regular file to be used as input. If no *file* operand is specified, the
 109579 *strings* utility shall read from the standard input.

109580 **STDIN**

109581 See the INPUT FILES section.

109582 **INPUT FILES**

109583 The input files named by the utility arguments or the standard input shall be regular files of any
 109584 format.

109585 **ENVIRONMENT VARIABLES**109586 The following environment variables shall affect the execution of *strings*:

109587 **LANG** Provide a default value for the internationalization variables that are unset or null.
 109588 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 109589 variables used to determine the values of locale categories.)

109590 **LC_ALL** If set to a non-empty string value, override the values of all the other
 109591 internationalization variables.

109592 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 109593 characters (for example, single-byte as opposed to multi-byte characters in
 109594 arguments and input files) and to identify printable strings.

109595 *LC_MESSAGES*
 109596 Determine the locale that should be used to affect the format and contents of
 109597 diagnostic messages written to standard error.

109598 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

109599 **ASYNCHRONOUS EVENTS**
 109600 Default.

109601 **STDOUT**
 109602 Strings found shall be written to the standard output, one per line.
 109603 When the `-t` option is not specified, the format of the output shall be:
 109604 `"%s", <string>`
 109605 With the `-t o` option, the format of the output shall be:
 109606 `"%o %s", <byte offset>, <string>`
 109607 With the `-t x` option, the format of the output shall be:
 109608 `"%x %s", <byte offset>, <string>`
 109609 With the `-t d` option, the format of the output shall be:
 109610 `"%d %s", <byte offset>, <string>`

109611 **STDERR**
 109612 The standard error shall be used only for diagnostic messages.

109613 **OUTPUT FILES**
 109614 None.

109615 **EXTENDED DESCRIPTION**
 109616 None.

109617 **EXIT STATUS**
 109618 The following exit values shall be returned:
 109619 0 Successful completion.
 109620 >0 An error occurred.

109621 **CONSEQUENCES OF ERRORS**
 109622 Default.

109623 **APPLICATION USAGE**
 109624 By default the data area (as opposed to the text, ```bss''`, or header areas) of a binary executable
 109625 file is scanned. Implementations document which areas are scanned.
 109626 Some historical implementations do not require NUL or `<newline>` terminators for strings to
 109627 permit those languages that do not use NUL as a string terminator to have their strings written.

109628 **EXAMPLES**
 109629 None.

109630 **RATIONALE**
 109631 Apart from rationalizing the option syntax and slight difficulties with object and executable
 109632 binary files, *strings* is specified to match historical practice closely. The `-a` and `-n` options were
 109633 introduced to replace the non-conforming `-` and `-number` options. These options are no longer
 109634 specified by POSIX.1-2017 but may be present in some implementations.

- 109635 The `-o` option historically means different things on different implementations. Some use it to
109636 mean “offset in decimal”, while others use it as “offset in octal”. Instead of trying to decide which
109637 way would be least objectionable, the `-t` option was added. It was originally named `-O` to mean
109638 “offset”, but was changed to `-t` to be consistent with *od*.
- 109639 The ISO C standard function *isprint()* is restricted to a domain of **unsigned char**. This volume of
109640 POSIX.1-2017 requires implementations to write strings as defined by the current locale.
- 109641 **FUTURE DIRECTIONS**
- 109642 None.
- 109643 **SEE ALSO**
- 109644 *localedef*, *nm*
- 109645 XBD Chapter 8 (on page 173), Section 12.2 (on page 216)
- 109646 **CHANGE HISTORY**
- 109647 First released in Issue 4.
- 109648 **Issue 6**
- 109649 This utility is marked as part of the User Portability Utilities option.
- 109650 The obsolescent SYNOPSIS is removed.
- 109651 The normative text is reworded to avoid use of the term “must” for application requirements.
- 109652 **Issue 7**
- 109653 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
109654 argument is ‘-’.
- 109655 The *strings* utility is moved from the User Portability Utilities option to the Base. User
109656 Portability Utilities is now an option for interactive utilities.
- 109657 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

109658 **NAME**109659 strip — remove unnecessary information from strippable files (**DEVELOPMENT**)109660 **SYNOPSIS**109661 SD strip *file...*109662 **DESCRIPTION**109663 XSI A strippable file is defined as a relocatable, object, or executable file. On XSI-conformant
109664 systems, a strippable file can also be an archive of object or relocatable files.109665 The *strip* utility shall remove from strippable files named by the *file* operands any information
109666 the implementor deems unnecessary for execution of those files. The nature of that information
109667 is unspecified. The effect of *strip* on object and executable files shall be similar to the use of the
109668 XSI **-s** option to *c99* or *fort77*. The effect of *strip* on an archive of object files shall be similar to the
109669 use of the **-s** option to *c99* or *fort77* for each object file in the archive.109670 **OPTIONS**

109671 None.

109672 **OPERANDS**

109673 The following operand shall be supported:

109674 *file* A pathname referring to a strippable file.109675 **STDIN**

109676 Not used.

109677 **INPUT FILES**109678 The input files shall be in the form of strippable files successfully produced by any compiler
109679 XSI defined by this volume of POSIX.1-2017 or produced by creating or updating an archive of such
109680 files using the *ar* utility.109681 **ENVIRONMENT VARIABLES**109682 The following environment variables shall affect the execution of *strip*:109683 *LANG* Provide a default value for the internationalization variables that are unset or null.
109684 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
109685 variables used to determine the values of locale categories.)109686 *LC_ALL* If set to a non-empty string value, override the values of all the other
109687 internationalization variables.109688 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
109689 characters (for example, single-byte as opposed to multi-byte characters in
109690 arguments).109691 *LC_MESSAGES*109692 Determine the locale that should be used to affect the format and contents of
109693 diagnostic messages written to standard error.109694 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.109695 **ASYNCHRONOUS EVENTS**

109696 Default.

109697 **STDOUT**

109698 Not used.

109699 STDERR

109700 The standard error shall be used only for diagnostic messages.

109701 OUTPUT FILES

109702 The *strip* utility shall produce strippable files of unspecified format.

109703 EXTENDED DESCRIPTION

109704 None.

109705 EXIT STATUS

109706 The following exit values shall be returned:

109707 0 Successful completion.

109708 >0 An error occurred.

109709 CONSEQUENCES OF ERRORS

109710 Default.

109711 APPLICATION USAGE

109712 None.

109713 EXAMPLES

109714 None.

109715 RATIONALE

109716 Historically, this utility has been used to remove the symbol table from a strippable file. It was
109717 included since it is known that the amount of symbolic information can amount to several
109718 megabytes; the ability to remove it in a portable manner was deemed important, especially for
109719 smaller systems.

109720 The behavior of *strip* on object and executable files is said to be the same as the `-s` option to a
109721 compiler. While the end result is essentially the same, it is not required to be identical.

109722 XSI-conformant systems support use of *strip* on archive files containing object files or relocatable
109723 files.

109724 FUTURE DIRECTIONS

109725 None.

109726 SEE ALSO

109727 [ar](#), [c99](#), [fort77](#)

109728 XBD [Chapter 8](#) (on page 173)

109729 CHANGE HISTORY

109730 First released in Issue 2.

109731 Issue 6

109732 This utility is marked as part of the Software Development Utilities option.

109733 Issue 7

109734 Austin Group Interpretation 1003.1-2001 #103 is applied.

109735 **NAME**

109736 stty ‡'set the options for a terminal

109737 **SYNOPSIS**

109738 stty [-a|-g]

109739 stty *operand*...109740 **DESCRIPTION**

109741 The *stty* utility shall set or report on terminal I/O characteristics for the device that is its
 109742 standard input. Without options or operands specified, it shall report the settings of certain
 109743 characteristics, usually those that differ from implementation-defined defaults. Otherwise, it
 109744 shall modify the terminal state according to the specified operands. Detailed information about
 109745 the modes listed in the first five groups below are described in XBD [Chapter 11](#) (on page 199).
 109746 Operands in the Combination Modes group (see [Combination Modes](#), on page 3269) are
 109747 implemented using operands in the previous groups. Some combinations of operands are
 109748 mutually-exclusive on some terminal types; the results of using such combinations are
 109749 unspecified.

109750 Typical implementations of this utility require a communications line configured to use the
 109751 **termios** interface defined in the System Interfaces volume of POSIX.1-2017. On systems where
 109752 none of these lines are available, and on lines not currently configured to support the **termios**
 109753 interface, some of the operands need not affect terminal characteristics.

109754 **OPTIONS**109755 The *stty* utility shall conform to XBD [Section 12.2](#) (on page 216).

109756 The following options shall be supported:

- 109757 **-a** Write to standard output all the current settings for the terminal.
- 109758 **-g** Write to standard output all the current settings in an unspecified form that can be
 109759 used as arguments to another invocation of the *stty* utility on the same system. The
 109760 form used shall not contain any characters that would require quoting to avoid
 109761 word expansion by the shell; see [Section 2.6](#) (on page 2353).

109762 **OPERANDS**

109763 The following operands shall be supported to set the terminal characteristics.

109764 **Control Modes**

109765 **parenb** (**-parenb**) Enable (disable) parity generation and detection. This shall have the effect of
 109766 setting (not setting) PARENB in the **termios** *c_flag* field, as defined in XBD
 109767 [Chapter 11](#) (on page 199).

109768 **parodd** (**-parodd**) Select odd (even) parity. This shall have the effect of setting (not setting)
 109769 PARODD in the **termios** *c_flag* field, as defined in XBD [Chapter 11](#) (on page
 109770 199).
 109771

109772 **cs5 cs6 cs7 cs8** Select character size, if possible. This shall have the effect of setting CS5, CS6,
 109773 CS7, and CS8, respectively, in the **termios** *c_flag* field, as defined in XBD
 109774 [Chapter 11](#) (on page 199).

109775 *number* Set terminal baud rate to the number given, if possible. If the baud rate is set
 109776 to zero, the modem control lines shall no longer be asserted. This shall have
 109777 the effect of setting the input and output **termios** baud rate values as defined
 109778 in XBD [Chapter 11](#) (on page 199).

- 109779 **ispeed** *number* Set terminal input baud rate to the number given, if possible. If the input baud rate is set to zero, the input baud rate shall be specified by the value of the output baud rate. This shall have the effect of setting the input **termios** baud rate values as defined in XBD [Chapter 11](#) (on page 199).
- 109780
109781
109782
- 109783 **ospeed** *number* Set terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the output **termios** baud rate values as defined in XBD [Chapter 11](#) (on page 199).
- 109784
109785
109786
- 109787 **hupcl** (**-hupcl**) Stop asserting modem control lines (do not stop asserting modem control lines) on last close. This shall have the effect of setting (not setting) HUPCL in the **termios** *c_cflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109788
109789
- 109790 **hup** (**-hup**) Equivalent to **hupcl**(**-hupcl**).
- 109791
109792 **cstopb** (**-cstopb**) Use two (one) stop bits per character. This shall have the effect of setting (not setting) CSTOPB in the **termios** *c_cflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109793
109794 **cread** (**-cread**) Enable (disable) the receiver. This shall have the effect of setting (not setting) CREAD in the **termios** *c_cflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109795
109796
- 109797 **clocal** (**-clocal**) Assume a line without (with) modem control. This shall have the effect of setting (not setting) CLOCAL in the **termios** *c_cflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109798
109799

109800 It is unspecified whether *stty* shall report an error if an attempt to set a Control Mode fails.

109801 **Input Modes**

- 109802 **ignbrk** (**-ignbrk**) Ignore (do not ignore) break on input. This shall have the effect of setting (not setting) IGNBRK in the **termios** *c_iflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109803
109804
- 109805 **brkint** (**-brkint**) Signal (do not signal) INTR on break. This shall have the effect of setting (not setting) BRKINT in the **termios** *c_iflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109806
109807
- 109808 **ignpar** (**-ignpar**) Ignore (do not ignore) bytes with parity errors. This shall have the effect of setting (not setting) IGNPAR in the **termios** *c_iflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109809
109810
- 109811 **parmrk** (**-parmrk**) Mark (do not mark) parity errors. This shall have the effect of setting (not setting) PARMRK in the **termios** *c_iflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109812
109813
109814
- 109815 **inpck** (**-inpck**) Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the **termios** *c_iflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109816
109817
- 109818 **istrip** (**-istrip**) Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the **termios** *c_iflag* field, as defined in XBD [Chapter 11](#) (on page 199).
- 109819
109820

109821	inlcr (-inlcr)	Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 199).
109822		
109823		
109824	igncr (-igncr)	Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 199).
109825		
109826		
109827	icrnl (-icrnl)	Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 199).
109828		
109829		
109830	ixon (-ixon)	Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 199).
109831		
109832		
109833		
109834	ixany (-ixany)	Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 199).
109835		
109836		
109837	ixoff (-ixoff)	Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 199).
109838		
109839		
109840		

Output Modes

109841		
109842	opost (-opost)	Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109843		
109844		
109845	XSI onlcr (-onlcr)	Map (do not map) NL to CR-NL on output. This shall have the effect of setting (not setting) ONLCR in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109846		
109847		
109848	XSI ocrnl (-ocrnl)	Map (do not map) CR to NL on output. This shall have the effect of setting (not setting) OCRNL in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109849		
109850		
109851	XSI onocr (-onocr)	Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109852		
109853		
109854	XSI onlret (-onlret)	The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109855		
109856		
109857	XSI ofill (-ofill)	Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109858		
109859		
109860	XSI ofdel (-ofdel)	Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109861		
109862		

109863	XSI	cr0 cr1 cr2 cr3	Select the style of delay for CRs. This shall have the effect of setting CRDLY to CR0, CR1, CR2, or CR3, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109864			
109865			
109866	XSI	nl0 nl1	Select the style of delay for NL. This shall have the effect of setting NLDLY to NL0 or NL1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109867			
109868			
109869	XSI	tab0 tab1 tab2 tab3	Select the style of delay for horizontal tabs. This shall have the effect of setting TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199). Note that TAB3 has the effect of expanding <tab> characters to <space> characters.
109870			
109871			
109872			
109873			
109874	XSI	tabs (-tabs)	Synonym for tab0 (tab3) .
109875	XSI	bs0 bs1	Select the style of delay for <backspace> characters. This shall have the effect of setting BSDLY to BS0 or BS1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109876			
109877			
109878	XSI	ff0 ff1	Select the style of delay for <form-feed> characters. This shall have the effect of setting FFDLY to FF0 or FF1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109879			
109880			
109881	XSI	vt0 vt1	Select the style of delay for <vertical-tab> characters. This shall have the effect of setting VTDLY to VT0 or VT1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 199).
109882			
109883			
109884		Local Modes	
109885		isig (-isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 199).
109886			
109887			
109888			
109889		icanon (-icanon)	Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 199).
109890			
109891			
109892		iexten (-iexten)	Enable (disable) any implementation-defined special control characters not currently controlled by icanon , isig , ixon , or ixoff . This shall have the effect of setting (not setting) IEXTEN in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 199).
109893			
109894			
109895			
109896		echo (-echo)	Echo back (do not echo back) every character typed. This shall have the effect of setting (not setting) ECHO in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 199).
109897			
109898			
109899		echoe (-echoe)	The ERASE character visually erases (does not erase) the last character in the current line from the display, if possible. This shall have the effect of setting (not setting) ECHOE in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 199).
109900			
109901			
109902			
109903		echok (-echok)	Echo (do not echo) NL after KILL character. This shall have the effect of setting (not setting) ECHOK in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 199).
109904			
109905			

- 109906 **echonl** (**-echonl**) Echo (do not echo) NL, even if **echo** is disabled. This shall have the effect of
 109907 setting (not setting) ECHONL in the **termios** *c_lflag* field, as defined in XBD
 109908 [Chapter 11](#) (on page 199).
- 109909 **noflsh** (**-noflsh**) Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of
 109910 setting (not setting) NOFLSH in the **termios** *c_lflag* field, as defined in XBD
 109911 [Chapter 11](#) (on page 199).
- 109912 **tostop** (**-tostop**) Send SIGTTOU for background output. This shall have the effect of setting
 109913 (not setting) TOSTOP in the **termios** *c_lflag* field, as defined in XBD [Chapter 11](#)
 109914 (on page 199).

109915 Special Control Character Assignments

- 109916 *<control>-character string*
 109917 Set *<control>-character* to *string*. If *<control>-character* is one of the character sequences in the
 109918 first column of the following table, the corresponding XBD [Chapter 11](#) (on page 199) control
 109919 character from the second column shall be recognized. This has the effect of setting the
 109920 corresponding element of the **termios** *c_cc* array (see XBD [Chapter 13](#) (on page 219),
 109921 **<termios.h>**).

109922 **Table 4-20** Control Character Names in *stty*

Control Character	c_cc Subscript	Description
eof	VEOF	EOF character
eol	VEOL	EOL character
erase	VERASE	ERASE character
intr	VINTR	INTR character
kill	VKILL	KILL character
quit	VQUIT	QUIT character
susp	VSUSP	SUSP character
start	VSTART	START character
stop	VSTOP	STOP character

- 109933 If *string* is a single character, the control character shall be set to that character. If *string* is the
 109934 two-character sequence "**^**-" or the string *undef*, the control character shall be set to
 109935 **_POSIX_VDISABLE**, if it is in effect for the device; if **_POSIX_VDISABLE** is not in effect for
 109936 the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character
 109937 sequence beginning with **<circumflex>** ('**^**'), and the second character is one of those listed
 109938 in the "**^c**" column of the following table, the control character shall be set to the
 109939 corresponding character value in the Value column of the table.

109940

Table 4-21 Circumflex Control Characters in *stty*

109941

109942

109943

109944

109945

109946

109947

109948

109949

109950

109951

109952

^c	Value	^c	Value	^c	Value
a, A	<SOH>	l, L	<FF>	w, W	<ETB>
b, B	<STX>	m, M	<CR>	x, X	<CAN>
c, C	<ETX>	n, N	<SO>	y, Y	
d, D	<EOT>	o, O	<SI>	z, Z	<SUB>
e, E	<ENQ>	p, P	<DLE>	[<ESC>
f, F	<ACK>	q, Q	<DC1>	\	<FS>
g, G	<BEL>	r, R	<DC2>]	<GS>
h, H	<BS>	s, S	<DC3>	^	<RS>
i, I	<HT>	t, T	<DC4>	_	<US>
j, J	<LF>	u, U	<NAK>	?	
k, K	<VT>	v, V	<SYN>		

109953

min number

109954

Set the value of MIN to *number*. MIN is used in non-canonical mode input processing (**icanon**).

109955

109956

time number

109957

Set the value of TIME to *number*. TIME is used in non-canonical mode input processing (**icanon**).

109958

109959

Combination Modes

109960

saved settings

109961

Set the current terminal characteristics to the saved settings produced by the **-g** option.

109962

evenp or parity

109963

Enable **parenb** and **cs7**; disable **parodd**.

109964

oddp

109965

Enable **parenb**, **cs7**, and **parodd**.

109966

-parity, -evenp, or -oddp

109967

Disable **parenb**, and set **cs8**.

109968 XSI

raw (-raw or cooked)

109969

Enable (disable) raw input and output. Raw mode shall be equivalent to setting:

109970

```
stty cs8 erase ^- kill ^- intr ^- \
quit ^- eof ^- eol ^- -post -inpck
```

109971

109972

nl (-nl)

109973

Disable (enable) **icrnl**. In addition, **-nl** unsets **inlcr** and **igncr**.

109974

ek Reset ERASE and KILL characters back to system defaults.

109975

sane

109976

Reset all modes to some reasonable, unspecified, values.

109977 **STDIN**

109978

Although no input is read from standard input, standard input shall be used to get the current terminal I/O characteristics and to set new terminal I/O characteristics.

109979

109980 **INPUT FILES**

109981 None.

109982 **ENVIRONMENT VARIABLES**109983 The following environment variables shall affect the execution of *stty*:

109984 *LANG* Provide a default value for the internationalization variables that are unset or null.
 109985 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 109986 variables used to determine the values of locale categories.)

109987 *LC_ALL* If set to a non-empty string value, override the values of all the other
 109988 internationalization variables.

109989 *LC_CTYPE* This variable determines the locale for the interpretation of sequences of bytes of
 109990 text data as characters (for example, single-byte as opposed to multi-byte
 109991 characters in arguments) and which characters are in the class **print**.

109992 *LC_MESSAGES*

109993 Determine the locale that should be used to affect the format and contents of
 109994 diagnostic messages written to standard error.

109995 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

109996 **ASYNCHRONOUS EVENTS**

109997 Default.

109998 **STDOUT**

109999 If operands are specified, no output shall be produced.

110000 If the **-g** option is specified, *stty* shall write to standard output the current settings in a form that
 110001 can be used as arguments to another instance of *stty* on the same system.

110002 If the **-a** option is specified, all of the information as described in the OPERANDS section shall
 110003 be written to standard output. Unless otherwise specified, this information shall be written as
 110004 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified
 110005 number of tokens per line. Additional information may be written.

110006 If no options or operands are specified, an unspecified subset of the information written for the
 110007 **-a** option shall be written.

110008 If speed information is written as part of the default output, or if the **-a** option is specified and if
 110009 the terminal input speed and output speed are the same, the speed information shall be written
 110010 as follows:

110011 "speed %d baud;", <speed>

110012 Otherwise, speeds shall be written as:

110013 "ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>

110014 In locales other than the POSIX locale, the word **baud** may be changed to something more
 110015 appropriate in those locales.

110016 If control characters are written as part of the default output, or if the **-a** option is specified,
 110017 control characters shall be written as:

110018 "%s = %s;", <control-character name>, <value>

110019 where <value> is either the character, or some visual representation of the character if it is non-
 110020 printable, or the string *undef* if the character is disabled.

110021 **STDERR**

110022 The standard error shall be used only for diagnostic messages.

110023 **OUTPUT FILES**

110024 None.

110025 **EXTENDED DESCRIPTION**

110026 None.

110027 **EXIT STATUS**

110028 The following exit values shall be returned:

110029 0 The terminal options were read or set successfully.

110030 >0 An error occurred.

110031 **CONSEQUENCES OF ERRORS**

110032 Default.

110033 **APPLICATION USAGE**

110034 The `-g` flag is designed to facilitate the saving and restoring of terminal state from the shell level.

110035 For example, a program may:

```
110036 saveterm="$(stty -g)"           # save terminal state
110037 stty (new settings)           # set new state
110038 ...                           # ...
110039 stty $saveterm                # restore terminal state
```

110040 Since the format is unspecified, the saved value is not portable across systems.

110041 Since the `-a` format is so loosely specified, scripts that save and restore terminal settings should use the `-g` option.

110043 **EXAMPLES**

110044 None.

110045 **RATIONALE**

110046 The original *stty* description was taken directly from System V and reflected the System V terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

110048 Output modes are specified only for XSI-conformant systems. All implementations are expected to provide *stty* operands corresponding to all of the output modes they support.

110050 The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the preferred ERASE and KILL characters. As an application programming utility, *stty* can be used within shell scripts to alter the terminal settings for the duration of the script.

110053 The **termios** section states that individual disabling of control characters is possible through the option `_POSIX_VDISABLE`. If enabled, two conventions currently exist for specifying this: System V uses "`^-`", and BSD uses *undef*. Both are accepted by *stty* in this volume of POSIX.1-2017. The other BSD convention of using the letter 'u' was rejected because it conflicts with the actual letter 'u', which is an acceptable value for a control character.

110058 Early proposals did not specify the mapping of "`^c`" to control characters because the control characters were not specified in the POSIX locale character set description file requirements. The control character set is now specified in XBD [Chapter 3](#) (on page 33), so the historical mapping is specified. Note that although the mapping corresponds to control-character key assignments on many terminals that use the ISO/IEC 646:1991 standard (or ASCII) character encodings, the mapping specified here is to the control characters, not their keyboard encodings.

- 110064 Since **termios** supports separate speeds for input and output, two new options were added to
110065 specify each distinctly.
- 110066 Some historical implementations use standard input to get and set terminal characteristics;
110067 others use standard output. Since input from a login TTY is usually restricted to the owner while
110068 output to a TTY is frequently open to anyone, using standard input provides fewer chances of
110069 accidentally (or maliciously) altering the terminal settings of other users. Using standard input
110070 also allows *stty -a* and *stty -g* output to be redirected for later use. Therefore, usage of standard
110071 input is required by this volume of POSIX.1-2017.
- 110072 **FUTURE DIRECTIONS**
- 110073 None.
- 110074 **SEE ALSO**
- 110075 [Chapter 2](#) (on page 2345)
- 110076 XBD [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 216), [<termios.h>](#)
- 110077 **CHANGE HISTORY**
- 110078 First released in Issue 2.
- 110079 **Issue 5**
- 110080 The description of **tabs** is clarified.
- 110081 The FUTURE DIRECTIONS section is added.
- 110082 **Issue 6**
- 110083 The LEGACY items **iucl(-iucl)**, **xcase**, **olcuc(-olcuc)**, **lcase(-lcase)**, and **LCASE(-LCASE)** are
110084 removed.
- 110085 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/37 is applied, applying IEEE PASC
110086 Interpretation 1003.2 #133, fixing an error in the OPERANDS section for the Combination Modes
110087 **nl(-nl)**.
- 110088 **Issue 7**
- 110089 Austin Group Interpretation 1003.1-2001 #144 is applied, moving functionality relating to the
110090 IXANY symbol from the XSI option to the Base.
- 110091 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 110092 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0189 [908] is applied.

110093 **NAME**110094 `tabs` ‡set terminal tabs110095 **SYNOPSIS**110096 XSI `tabs [-n|-a|-a2|-c|-c2|-c3|-f|-p|-s|-u] [-T type]`110097 `tabs [-T type] n[[sep[+]n]...]`110098 **DESCRIPTION**

110099 The `tabs` utility shall display a series of characters that first clears the hardware terminal tab
 110100 XSI settings and then initializes the tab stops at the specified positions and optionally adjusts the
 110101 margin.

110102 The phrase “tab-stop position *N*” shall be taken to mean that, from the start of a line of output,
 110103 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position
 110104 on that line. The maximum number of tab stops allowed is terminal-dependent.

110105 It need not be possible to implement `tabs` on certain terminals. If the terminal type obtained from
 110106 the `TERM` environment variable or `-T` option represents such a terminal, an appropriate
 110107 diagnostic message shall be written to standard error and `tabs` shall exit with a status greater
 110108 than zero.

110109 **OPTIONS**

110110 XSI The `tabs` utility shall conform to XBD Section 12.2 (on page 216), except for various extensions:
 110111 the options `-a2`, `-c2`, and `-c3` are multi-character.

110112 The following options shall be supported:

110113 `-n` Specify repetitive tab stops separated by a uniform number of column positions, *n*,
 110114 where *n* is a single-digit decimal number. The default usage of `tabs` with no
 110115 arguments shall be equivalent to `tabs -8`. When `-0` is used, the tab stops shall be
 110116 cleared and no new ones set.

110117 XSI `-a` 1,10,16,36,72
 110118 Assembler, applicable to some mainframes.

110119 XSI `-a2` 1,10,16,40,72
 110120 Assembler, applicable to some mainframes.

110121 XSI `-c` 1,8,12,16,20,55
 110122 COBOL, normal format.

110123 XSI `-c2` 1,6,10,14,49
 110124 COBOL, compact format (columns 1 to 6 omitted).

110125 XSI `-c3` 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
 110126 COBOL compact format (columns 1 to 6 omitted), with more tabs than `-c2`.

110127 XSI `-f` 1,7,11,15,19,23
 110128 FORTRAN

110129 XSI `-p` 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
 110130 PL/1

110131 XSI `-s` 1,10,55
 110132 SNOBOL

110133 XSI `-u` 1,12,20,44
 110134 Assembler, applicable to some mainframes.

110135 -T *type* Indicate the type of terminal. If this option is not supplied and the *TERM* variable
 110136 is unset or null, an unspecified default terminal type shall be used. The setting of
 110137 *type* shall take precedence over the value in *TERM*.

110138 **OPERANDS**

110139 The following operand shall be supported:

110140 *n*[[*sep*[+]*n*]...] A single command line argument that consists of one or more tab-stop values (*n*)
 110141 separated by a separator character (*sep*) which is either a <comma> or a <blank>
 110142 character. The application shall ensure that the tab-stop values are positive decimal
 110143 integers in strictly ascending order. If any tab-stop value (except the first one) is
 110144 preceded by a <plus-sign>, it is taken as an increment to be added to the previous
 110145 value. For example, the tab lists 1,10,20,30 and "1 10 +10 +10" are considered
 110146 to be identical.

110147 **STDIN**

110148 Not used.

110149 **INPUT FILES**

110150 None.

110151 **ENVIRONMENT VARIABLES**

110152 The following environment variables shall affect the execution of *tabs*:

110153 *LANG* Provide a default value for the internationalization variables that are unset or null.
 110154 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 110155 variables used to determine the values of locale categories.)

110156 *LC_ALL* If set to a non-empty string value, override the values of all the other
 110157 internationalization variables.

110158 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 110159 characters (for example, single-byte as opposed to multi-byte characters in
 110160 arguments).

110161 *LC_MESSAGES*

110162 Determine the locale that should be used to affect the format and contents of
 110163 diagnostic messages written to standard error.

110164 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

110165 *TERM* Determine the terminal type. If this variable is unset or null, and if the -T option is
 110166 not specified, an unspecified default terminal type shall be used.

110167 **ASYNCHRONOUS EVENTS**

110168 Default.

110169 **STDOUT**

110170 If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be
 110171 written to standard output in an unspecified format. If standard output is not a terminal,
 110172 undefined results occur.

110173 **STDERR**

110174 The standard error shall be used only for diagnostic messages.

110175 **OUTPUT FILES**

110176 None.

110177 **EXTENDED DESCRIPTION**

110178 None.

110179 **EXIT STATUS**

110180 The following exit values shall be returned:

110181 0 Successful completion.

110182 >0 An error occurred.

110183 **CONSEQUENCES OF ERRORS**

110184 Default.

110185 **APPLICATION USAGE**110186 This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

110187 This utility is not recommended for application use.

110188 Some integrated display units might not have escape sequences to set tab stops, but may be set
 110189 by internal system calls. On these terminals, *tabs* works if standard output is directed to the
 110190 terminal; if output is directed to another file, however, *tabs* fails.

110191 **EXAMPLES**

110192 None.

110193 **RATIONALE**

110194 Consideration was given to having the *tput* utility handle all of the functions described in *tabs*.
 110195 However, the separate *tabs* utility was retained because it seems more intuitive to use a
 110196 command named *tabs* than *tput* with a new option. The *tput* utility does not support setting or
 110197 clearing tabs, and no known historical version of *tabs* supports the capability of setting arbitrary
 110198 tab stops.

110199 The System V *tabs* interface is very complex; the version in this volume of POSIX.1-2017 has a
 110200 reduced feature list, but many of the features omitted were restored as part of the XSI option
 110201 even though the supported languages and coding styles are primarily historical.

110202 There was considerable sentiment for specifying only a means of resetting the tabs back to a
 110203 known state ~~the~~ presumably the "standard" of tabs every eight positions. The following features
 110204 were omitted:

110205 Setting tab stops via the first line in a file, using *--file*. Since even the SVID has no
 110206 complete explanation of this feature, it is doubtful that it is in widespread use.

110207 In an early proposal, a *-t tablist* option was added for consistency with *expand*; this was later
 110208 removed when inconsistencies with the historical list of tabs were identified.

110209 Consideration was given to adding a *-p* option that would output the current tab settings so
 110210 that they could be saved and then later restored. This was not accepted because querying the tab
 110211 stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be
 110212 supported on a wide range of terminals.

110213 **FUTURE DIRECTIONS**

110214 None.

110215 **SEE ALSO**110216 *expand*, *stty*, *tput*, *unexpand*

110217 XBD Chapter 8 (on page 173), Section 12.2 (on page 216)

110218 **CHANGE HISTORY**

110219 First released in Issue 2.

110220 **Issue 6**

110221 This utility is marked as part of the User Portability Utilities option.

110222 The normative text is reworded to avoid use of the term “must” for application requirements.

110223 **Issue 7**

110224 The *tabs* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

110226 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

110227 The SYNOPSIS and OPERANDS sections are updated.

110228 **NAME**

110229 tail ‡copy the last part of a file

110230 **SYNOPSIS**110231 tail [-f] [-c *number*|-n *number*] [*file*]110232 **DESCRIPTION**110233 The *tail* utility shall copy its input file to the standard output beginning at a designated place.

110234 Copying shall begin at the point in the file indicated by the `-c number` or `-n number` options. The
 110235 option-argument *number* shall be counted in units of lines or bytes, according to the options `-n`
 110236 and `-c`. Both line and byte counts start from 1.

110237 Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in
 110238 length. Such a buffer, if any, shall be no smaller than `{LINE_MAX}*10` bytes.

110239 **OPTIONS**

110240 The *tail* utility shall conform to XBD [Section 12.2](#) (on page 216), except that '+' may be
 110241 recognized as an option delimiter as well as '-'.

110242 The following options shall be supported:

110243 `-c number` The application shall ensure that the *number* option-argument is a decimal integer,
 110244 optionally including a sign. The sign shall affect the location in the file, measured
 110245 in bytes, to begin the copying:

Sign	Copying Starts
+	Relative to the beginning of the file.
-	Relative to the end of the file.
<i>none</i>	Relative to the end of the file.

110250 The application shall ensure that if the sign of the *number* option-argument is '+',
 110251 the *number* option-argument is a non-zero decimal integer.

110252 The origin for counting shall be 1; that is, `-c +1` represents the first byte of the file,
 110253 `-c -1` the last.

110254 `-f` If the input file is a regular file or if the *file* operand specifies a FIFO, do not
 110255 terminate after the last line of the input file has been copied, but read and copy
 110256 further bytes from the input file when they become available. If no *file* operand is
 110257 specified and standard input is a pipe or FIFO, the `-f` option shall be ignored. If the
 110258 input file is not a FIFO, pipe, or regular file, it is unspecified whether or not the `-f`
 110259 option shall be ignored.

110260 `-n number` This option shall be equivalent to `-c number`, except the starting location in the file
 110261 shall be measured in lines instead of bytes. The origin for counting shall be 1; that
 110262 is, `-n +1` represents the first line of the file, `-n -1` the last.

110263 If neither `-c` nor `-n` is specified, `-n 10` shall be assumed.

110264 **OPERANDS**

110265 The following operand shall be supported:

110266 *file* A pathname of an input file. If no *file* operand is specified, the standard input shall
 110267 be used.

110268 **STDIN**

110269 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*
110270 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
110271 the standard input shall not be used. See the INPUT FILES section.

110272 **INPUT FILES**

110273 If the -c option is specified, the input file can contain arbitrary data; otherwise, the input file
110274 shall be a text file.

110275 **ENVIRONMENT VARIABLES**

110276 The following environment variables shall affect the execution of *tail*:

110277 *LANG* Provide a default value for the internationalization variables that are unset or null.
110278 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
110279 variables used to determine the values of locale categories.)

110280 *LC_ALL* If set to a non-empty string value, override the values of all the other
110281 internationalization variables.

110282 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
110283 characters (for example, single-byte as opposed to multi-byte characters in
110284 arguments and input files).

110285 *LC_MESSAGES*

110286 Determine the locale that should be used to affect the format and contents of
110287 diagnostic messages written to standard error.

110288 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

110289 **ASYNCHRONOUS EVENTS**

110290 Default.

110291 **STDOUT**

110292 The designated portion of the input file shall be written to standard output.

110293 **STDERR**

110294 The standard error shall be used only for diagnostic messages.

110295 **OUTPUT FILES**

110296 None.

110297 **EXTENDED DESCRIPTION**

110298 None.

110299 **EXIT STATUS**

110300 The following exit values shall be returned:

110301 0 Successful completion.

110302 >0 An error occurred.

110303 **CONSEQUENCES OF ERRORS**

110304 Default.

110305 **APPLICATION USAGE**

110306 The `-c` option should be used with caution when the input is a text file containing multi-byte
 110307 characters; it may produce output that does not start on a character boundary.

110308 Although the input file to *tail* can be any type, the results might not be what would be expected
 110309 on some character special device files or on file types not described by the System Interfaces
 110310 volume of POSIX.1-2017. Since this volume of POSIX.1-2017 does not specify the block size used
 110311 when doing input, *tail* need not read all of the data from devices that only perform block
 110312 transfers.

110313 When using *tail* to process pathnames, and the `-c` option is not specified, it is recommended that
 110314 `LC_ALL`, or at least `LC_CTYPE` and `LC_COLLATE`, are set to `POSIX` or `C` in the environment,
 110315 since pathnames can contain byte sequences that do not form valid characters in some locales, in
 110316 which case the utility's behavior would be undefined. In the POSIX locale each byte is a valid
 110317 single-byte character, and therefore this problem is avoided.

110318 **EXAMPLES**

110319 The `-f` option can be used to monitor the growth of a file that is being written by some other
 110320 process. For example, the command:

```
110321 tail -f fred
```

110322 prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between
 110323 the time *tail* is initiated and killed. As another example, the command:

```
110324 tail -f -c 15 fred
```

110325 prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between
 110326 the time *tail* is initiated and killed.

110327 **RATIONALE**

110328 This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The
 110329 historical `-b` option was omitted because of the general non-portability of block-sized units of
 110330 text. The `-c` option historically meant "characters", but this volume of POSIX.1-2017 indicates
 110331 that it means "bytes". This was selected to allow reasonable implementations when multi-byte
 110332 characters are possible; it was not named `-b` to avoid confusion with the historical `-b`.

110333 The origin of counting both lines and bytes is 1, matching all widespread historical
 110334 implementations. Hence *tail* `-n +0` is not conforming usage because it attempts to output line
 110335 zero; but note that *tail* `-n 0` does conform, and outputs nothing.

110336 Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
110337 tail -[number] [b|c|l] [f] [file]
```

```
110338 tail +[number] [b|c|l] [f] [file]
```

110339 These forms are no longer specified by POSIX.1-2017, but may be present in some
 110340 implementations.

110341 The restriction on the internal buffer is a compromise between the historical System V
 110342 implementation of 4 096 bytes and the BSD 32 768 bytes.

110343 The `-f` option has been implemented as a loop that sleeps for 1 second and copies any bytes that
 110344 are available. This is sufficient, but if more efficient methods of determining when new data are
 110345 available are developed, implementations are encouraged to use them.

110346 Historical documentation indicates that *tail* ignores the `-f` option if the input file is a pipe (pipe
 110347 and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System
 110348 V-based systems, this was true when input was taken from standard input, but it did not ignore

110349 the `-f` flag if a FIFO was named as the *file* operand. Since the `-f` option is not useful on pipes and
110350 all historical implementations ignore `-f` if no *file* operand is specified and standard input is a
110351 pipe, this volume of POSIX.1-2017 requires this behavior. However, since the `-f` option is useful
110352 on a FIFO, this volume of POSIX.1-2017 also requires that if a FIFO is named, the `-f` option shall
110353 not be ignored. Earlier versions of this standard did not state any requirement for the case where
110354 no *file* operand is specified and standard input is a FIFO. The standard has been updated to
110355 reflect current practice which is to treat this case the same as a pipe on standard input. Although
110356 historical behavior does not ignore the `-f` option for other file types, this is unspecified so that
110357 implementations are allowed to ignore the `-f` option if it is known that the file cannot be
110358 extended.

110359 **FUTURE DIRECTIONS**

110360 None.

110361 **SEE ALSO**

110362 *head*

110363 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

110364 **CHANGE HISTORY**

110365 First released in Issue 2.

110366 **Issue 6**

110367 The obsolescent SYNOPSIS lines and associated text are removed.

110368 The normative text is reworded to avoid use of the term “must” for application requirements.

110369 **Issue 7**

110370 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized
110371 as an option delimiter in the OPTIONS section.

110372 Austin Group Interpretation 1003.1-2001 #092 is applied.

110373 Austin Group Interpretation 1003.1-2001 #100 is applied, adding the requirement on applications
110374 that if the sign of the option-argument *number* is '+', the *number* option-argument is a non-zero
110375 decimal integer.

110376 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

110377 SD5-XCU-ERN-114 is applied, updating the OPTIONS section (the `-f` option).

110378 SD5-XCU-ERN-149 is applied.

110379 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0190 [663] is applied.

110380 **NAME**

110381 talk ‡talk to another user

110382 **SYNOPSIS**110383 UP `talk address [terminal]`110384 **DESCRIPTION**110385 The *talk* utility is a two-way, screen-oriented communication program.110386 When first invoked, *talk* shall send a message similar to:

```
110387 Message from <unspecified string>
110388 talk: connection requested by your_address
110389 talk: respond with: talk your_address
```

110390 to the specified *address*. At this point, the recipient of the message can reply by typing:110391 `talk your_address`110392 Once communication is established, the two parties can type simultaneously, with their output
110393 displayed in separate regions of the screen. Characters shall be processed as follows:

110394 Typing the <alert> character shall alert the recipient's terminal.

110395 Typing <control>-L shall cause the sender's screen regions to be refreshed.

110396 Typing the erase and kill characters shall affect the sender's terminal in the manner
110397 described by the **termios** interface in XBD [Chapter 11](#) (on page 199).110398 Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the
110399 *talk* session has been terminated on one side, the other side of the *talk* session shall be
110400 notified that the *talk* session has been terminated and shall be able to do nothing except
110401 exit.110402 Typing characters from *LC_CTYPE* classifications **print** or **space** shall cause those
110403 characters to be sent to the recipient's terminal.110404 When and only when the *stty* **ixten** local mode is enabled, the existence and processing of
110405 additional special control characters and multi-byte or single-byte functions shall be
110406 implementation-defined.110407 Typing other non-printable characters shall cause implementation-defined sequences of
110408 printable characters to be sent to the recipient's terminal.110409 Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility.
110410 However, a user's privilege may further constrain the domain of accessibility of other users'
110411 terminals. The *talk* utility shall fail when the user lacks appropriate privileges to perform the
110412 requested action.110413 Certain block-mode terminals do not have all the capabilities necessary to support the
110414 simultaneous exchange of messages required for *talk*. When this type of exchange cannot be
110415 supported on such terminals, the implementation may support an exchange with reduced levels
110416 of simultaneous interaction or it may report an error describing the terminal-related deficiency.110417 **OPTIONS**

110418 None.

110419 **OPERANDS**

110420 The following operands shall be supported:

110421 *address* The recipient of the *talk* session. One form of *address* is the *<user name>*, as returned
 110422 by the *who* utility. Other address formats and how they are handled are
 110423 unspecified.

110424 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to
 110425 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message
 110426 shall be displayed on one or more accessible terminals in use by the recipient. The
 110427 format of *terminal* shall be the same as that returned by the *who* utility.

110428 **STDIN**

110429 Characters read from standard input shall be copied to the recipient's terminal in an unspecified
 110430 manner. If standard input is not a terminal, *talk* shall write a diagnostic message and exit with a
 110431 non-zero status.

110432 **INPUT FILES**

110433 None.

110434 **ENVIRONMENT VARIABLES**

110435 The following environment variables shall affect the execution of *talk*:

110436 *LANG* Provide a default value for the internationalization variables that are unset or null.
 110437 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 110438 variables used to determine the values of locale categories.)

110439 *LC_ALL* If set to a non-empty string value, override the values of all the other
 110440 internationalization variables.

110441 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 110442 characters (for example, single-byte as opposed to multi-byte characters in
 110443 arguments and input files). If the recipient's locale does not use an *LC_CTYPE*
 110444 equivalent to the sender's, the results are undefined.

110445 *LC_MESSAGES*

110446 Determine the locale that should be used to affect the format and contents of
 110447 diagnostic messages written to standard error and informative messages written to
 110448 standard output.

110449 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

110450 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,
 110451 an unspecified default terminal type shall be used.

110452 **ASYNCHRONOUS EVENTS**

110453 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero
 110454 status. It shall take the standard action for all other signals.

110455 **STDOUT**

110456 If standard output is a terminal, characters copied from the recipient's standard input may be
 110457 written to standard output. Standard output also may be used for diagnostic messages. If
 110458 standard output is not a terminal, *talk* shall exit with a non-zero status.

110459 **STDERR**

110460 None.

110461 **OUTPUT FILES**

110462 None.

110463 **EXTENDED DESCRIPTION**

110464 None.

110465 **EXIT STATUS**

110466 The following exit values shall be returned:

110467 0 Successful completion.

110468 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.110469 **CONSEQUENCES OF ERRORS**

110470 Default.

110471 **APPLICATION USAGE**

110472 Because the handling of non-printable, non-`<space>` characters is tied to the *stty* description of
 110473 **ixten**, implementation extensions within the terminal driver can be accessed. For example,
 110474 some implementations provide line editing functions with certain control character sequences.

110475 **EXAMPLES**

110476 None.

110477 **RATIONALE**

110478 The *write* utility was included in this volume of POSIX.1-2017 since it can be implemented on all
 110479 terminal types. The *talk* utility, which cannot be implemented on certain terminals, was
 110480 considered to be a “better” communications interface. Both of these programs are in widespread
 110481 use on historical implementations. Therefore, both utilities have been specified.

110482 All references to networking abilities (*talking* to a user on another system) were removed as
 110483 being outside the scope of this volume of POSIX.1-2017.

110484 Historical BSD and System V versions of *talk* terminate both of the conversations when either
 110485 user breaks out of the session. This can lead to adverse consequences if a user unwittingly
 110486 continues to enter text that is interpreted by the shell when the other terminates the session.
 110487 Therefore, the version of *talk* specified by this volume of POSIX.1-2017 requires both users to
 110488 terminate their end of the session explicitly.

110489 Only messages sent to the terminal of the invoking user can be internationalized in any way:

110490 The original “Message from `<unspecified string> ...`” message sent to the terminal of the
 110491 recipient cannot be internationalized because the environment of the recipient is as yet
 110492 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.

110493 Subsequent communication between the two parties cannot be internationalized because
 110494 the two parties may specify different languages in their environment (and non-portable
 110495 characters cannot be mapped from one language to another).

110496 Neither party can be required to communicate in a language other than C and/or the one
 110497 specified by their environment because unavailable terminal hardware support (for
 110498 example, fonts) may be required.

110499 The text in the STDOUT section reflects the usage of the verb “display” in this section; some *talk*
 110500 implementations actually use standard output to write to the terminal, but this volume of
 110501 POSIX.1-2017 does not require that to be the case.

110502 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
 110503 require that they all use or accept the same format.

110504 The handling of non-printable characters is partially implementation-defined because the details
110505 of mapping them to printable sequences is not needed by the user. Historical implementations,
110506 for security reasons, disallow the transmission of non-printable characters that may send
110507 commands to the other terminal.

110508 **FUTURE DIRECTIONS**

110509 None.

110510 **SEE ALSO**

110511 *mesg, stty, who, write*

110512 XBD [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199)

110513 **CHANGE HISTORY**

110514 First released in Issue 4.

110515 **Issue 6**

110516 This utility is marked as part of the User Portability Utilities option.

110517 **NAME**

110518 tee ‡duplicate standard input

110519 **SYNOPSIS**110520 tee [-ai] [*file...*]110521 **DESCRIPTION**110522 The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.110523 The *tee* utility shall not buffer output.110524 If the **-a** option is not specified, output files shall be written (see [Section 1.1.1.4](#) (on page 2328)).110525 **OPTIONS**110526 The *tee* utility shall conform to XBD [Section 12.2](#) (on page 216).

110527 The following options shall be supported:

110528 **-a** Append the output to the files.110529 **-i** Ignore the SIGINT signal.110530 **OPERANDS**

110531 The following operands shall be supported:

110532 *file* A pathname of an output file. If a *file* operand is '-', it shall refer to a file named
 110533 -; implementations shall not treat it as meaning standard output. Processing of at
 110534 least 13 *file* operands shall be supported.

110535 **STDIN**

110536 The standard input can be of any type.

110537 **INPUT FILES**

110538 None.

110539 **ENVIRONMENT VARIABLES**110540 The following environment variables shall affect the execution of *tee*:

110541 *LANG* Provide a default value for the internationalization variables that are unset or null.
 110542 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 110543 variables used to determine the values of locale categories.)

110544 *LC_ALL* If set to a non-empty string value, override the values of all the other
 110545 internationalization variables.

110546 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 110547 characters (for example, single-byte as opposed to multi-byte characters in
 110548 arguments).

110549 *LC_MESSAGES*

110550 Determine the locale that should be used to affect the format and contents of
 110551 diagnostic messages written to standard error.

110552 *XSHELL* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

110553 **ASYNCHRONOUS EVENTS**110554 Default, except that if the **-i** option was specified, SIGINT shall be ignored.110555 **STDOUT**

110556 The standard output shall be a copy of the standard input.

110557 STDERR

110558 The standard error shall be used only for diagnostic messages.

110559 OUTPUT FILES

110560 If any *file* operands are specified, the standard input shall be copied to each named file.

110561 EXTENDED DESCRIPTION

110562 None.

110563 EXIT STATUS

110564 The following exit values shall be returned:

110565 0 The standard input was successfully copied to all output files.

110566 >0 An error occurred.

110567 CONSEQUENCES OF ERRORS

110568 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file*
110569 operands and standard output shall continue, but the exit status shall be non-zero. Otherwise,
110570 the default actions specified in [Section 1.4](#) (on page 2336) apply.

110571 APPLICATION USAGE

110572 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

110573 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

110574 EXAMPLES

110575 Save an unsorted intermediate form of the data in a pipeline:

110576 `... | tee unsorted | sort > sorted`

110577 RATIONALE

110578 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or
110579 line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

110580 It should be noted that early versions of BSD ignore any invalid options and accept a single '-'
110581 as an alternative to -i. They also print a message if unable to open a file:

110582 `"tee: cannot access %s\n", <pathname>`

110583 Historical implementations ignore write errors. This is explicitly not permitted by this volume of
110584 POSIX.1-2017.

110585 Some historical implementations use O_APPEND when providing append mode; others use the
110586 *lseek()* function to seek to the end-of-file after opening the file without O_APPEND. This volume
110587 of POSIX.1-2017 requires functionality equivalent to using O_APPEND; see [Section 1.1.1.4](#) (on
110588 page 2328).

110589 FUTURE DIRECTIONS

110590 None.

110591 SEE ALSO

110592 [Chapter 1](#) (on page 2327), *cat*

110593 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

110594 XSH *lseek()*

110595 **CHANGE HISTORY**

110596 First released in Issue 2.

110597 **Issue 6**

110598 IEEE PASC Interpretation 1003.2 #168 is applied.

110599 **Issue 7**

110600 Austin Group Interpretation 1003.1-2001 #092 is applied.

110601 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

110602 **NAME**

110603 test — evaluate expression

110604 **SYNOPSIS**110605 test [*expression*]110606 [[*expression*]]110607 **DESCRIPTION**

110608 The *test* utility shall evaluate the *expression* and indicate the result of the evaluation by its exit
 110609 status. An exit status of zero indicates that the expression evaluated as true and an exit status of
 110610 1 indicates that the expression evaluated as false.

110611 In the second form of the utility, where the utility name used is *[* rather than *test*, the application
 110612 shall ensure that the closing square bracket is a separate argument. The *test* and *[* utilities may be
 110613 implemented as a single linked utility which examines the basename of the zeroth command
 110614 line argument to determine whether to behave as the *test* or *[* variant. Applications using the
 110615 *exec()* family of functions to execute these utilities shall ensure that the argument passed in *arg0*
 110616 or *argv[0]* is ' [' when executing the *[* utility and has a basename of "test" when executing the
 110617 *test* utility.

110618 **OPTIONS**

110619 The *test* utility shall not recognize the "--" argument in the manner specified by Guideline 10 in
 110620 XBD [Section 12.2](#) (on page 216).

110621 No options shall be supported.

110622 **OPERANDS**

110623 The application shall ensure that all operators and elements of primaries are presented as
 110624 separate arguments to the *test* utility.

110625 The following primaries can be used to construct *expression*:

110626 **-b** *pathname* True if *pathname* resolves to an existing directory entry for a block special file. False
 110627 if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
 110628 for a file that is not a block special file.

110629 **-c** *pathname* True if *pathname* resolves to an existing directory entry for a character special file.
 110630 False if *pathname* cannot be resolved, or if *pathname* resolves to an existing directory
 110631 entry for a file that is not a character special file.

110632 **-d** *pathname* True if *pathname* resolves to an existing directory entry for a directory. False if
 110633 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
 110634 for a file that is not a directory.

110635 **-e** *pathname* True if *pathname* resolves to an existing directory entry. False if *pathname* cannot be
 110636 resolved.

110637 **-f** *pathname* True if *pathname* resolves to an existing directory entry for a regular file. False if
 110638 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
 110639 for a file that is not a regular file.

110640 **-g** *pathname* True if *pathname* resolves to an existing directory entry for a file that has its set-
 110641 group-ID flag set. False if *pathname* cannot be resolved, or if *pathname* resolves to an
 110642 existing directory entry for a file that does not have its set-group-ID flag set.

110643 **-h** *pathname* True if *pathname* resolves to an existing directory entry for a symbolic link. False if
 110644 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
 110645 for a file that is not a symbolic link. If the final component of *pathname* is a
 110646 symbolic link, that symbolic link is not followed.

- 110647 **-L** *pathname* True if *pathname* resolves to an existing directory entry for a symbolic link. False if
 110648 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
 110649 for a file that is not a symbolic link. If the final component of *pathname* is a
 110650 symbolic link, that symbolic link is not followed.
- 110651 **-n** *string* True if the length of *string* is non-zero; otherwise, false.
- 110652 **-p** *pathname* True if *pathname* resolves to an existing directory entry for a FIFO. False if *pathname*
 110653 cannot be resolved, or if *pathname* resolves to an existing directory entry for a file
 110654 that is not a FIFO.
- 110655 **-r** *pathname* True if *pathname* resolves to an existing directory entry for a file for which
 110656 permission to read from the file will be granted, as defined in [Section 1.1.1.4](#) (on
 110657 page 2328). False if *pathname* cannot be resolved, or if *pathname* resolves to an
 110658 existing directory entry for a file for which permission to read from the file will not
 110659 be granted.
- 110660 **-S** *pathname* True if *pathname* resolves to an existing directory entry for a socket. False if
 110661 *pathname* cannot be resolved, or if *pathname* resolves to an existing directory entry
 110662 for a file that is not a socket.
- 110663 **-s** *pathname* True if *pathname* resolves to an existing directory entry for a file that has a size
 110664 greater than zero. False if *pathname* cannot be resolved, or if *pathname* resolves to an
 110665 existing directory entry for a file that does not have a size greater than zero.
- 110666 **-t** *file_descriptor* True if file descriptor number *file_descriptor* is open and is associated with a
 110667 terminal. False if *file_descriptor* is not a valid file descriptor number, or if file
 110668 descriptor number *file_descriptor* is not open, or if it is open but is not associated
 110669 with a terminal.
- 110671 **-u** *pathname* True if *pathname* resolves to an existing directory entry for a file that has its set-
 110672 user-ID flag set. False if *pathname* cannot be resolved, or if *pathname* resolves to an
 110673 existing directory entry for a file that does not have its set-user-ID flag set.
- 110674 **-w** *pathname* True if *pathname* resolves to an existing directory entry for a file for which
 110675 permission to write to the file will be granted, as defined in [Section 1.1.1.4](#) (on page
 110676 2328). False if *pathname* cannot be resolved, or if *pathname* resolves to an existing
 110677 directory entry for a file for which permission to write to the file will not be
 110678 granted.
- 110679 **-x** *pathname* True if *pathname* resolves to an existing directory entry for a file for which
 110680 permission to execute the file (or search it, if it is a directory) will be granted, as
 110681 defined in [Section 1.1.1.4](#) (on page 2328). False if *pathname* cannot be resolved, or if
 110682 *pathname* resolves to an existing directory entry for a file for which permission to
 110683 execute (or search) the file will not be granted.
- 110684 **-z** *string* True if the length of string *string* is zero; otherwise, false.
- 110685 *string* True if the string *string* is not the null string; otherwise, false.
- 110686 *s1* = *s2* True if the strings *s1* and *s2* are identical; otherwise, false.
- 110687 *s1* != *s2* True if the strings *s1* and *s2* are not identical; otherwise, false.
- 110688 *n1* -eq *n2* True if the integers *n1* and *n2* are algebraically equal; otherwise, false.

- 110689 *n1 -ne n2* True if the integers *n1* and *n2* are not algebraically equal; otherwise, false.
- 110690 *n1 -gt n2* True if the integer *n1* is algebraically greater than the integer *n2*; otherwise, false.
- 110691 *n1 -ge n2* True if the integer *n1* is algebraically greater than or equal to the integer *n2*;
110692 otherwise, false.
- 110693 *n1 -lt n2* True if the integer *n1* is algebraically less than the integer *n2*; otherwise, false.
- 110694 *n1 -le n2* True if the integer *n1* is algebraically less than or equal to the integer *n2*; otherwise,
110695 false.
- 110696 OB XSI *expression1 -a expression2*
110697 True if both *expression1* and *expression2* are true; otherwise, false. The *-a* binary
110698 primary is left associative. It has a higher precedence than *-o*.
- 110699 OB XSI *expression1 -o expression2*
110700 True if either *expression1* or *expression2* is true; otherwise, false. The *-o* binary
110701 primary is left associative.
- 110702 With the exception of the *-h pathname* and *-L pathname* primaries, if a *pathname* argument is a
110703 symbolic link, *test* shall evaluate the expression by resolving the symbolic link and using the file
110704 referenced by the link.
- 110705 These primaries can be combined with the following operators:
- 110706 *! expression* True if *expression* is false. False if *expression* is true.
- 110707 OB XSI *(expression)* True if *expression* is true. False if *expression* is false. The parentheses can be used to
110708 alter the normal precedence and associativity.
- 110709 The primaries with two elements of the form:
- 110710 *-primary_operator primary_operand*
- 110711 are known as *unary primaries*. The primaries with three elements in either of the two forms:
- 110712 *primary_operand -primary_operator primary_operand*
- 110713 *primary_operand primary_operator primary_operand*
- 110714 are known as *binary primaries*. Additional implementation-defined operators and
110715 *primary_operators* may be provided by implementations. They shall be of the form *-operator*
110716 where the first character of *operator* is not a digit.
- 110717 The algorithm for determining the precedence of the operators and the return value that shall be
110718 generated is based on the number of arguments presented to *test*. (However, when using the
110719 "[...]" form, the <right-square-bracket> final argument shall not be counted in this
110720 algorithm.)
- 110721 In the following list, \$1, \$2, \$3, and \$4 represent the arguments presented to *test*:
- 110722 0 arguments: Exit false (1).
- 110723 1 argument: Exit true (0) if \$1 is not null; otherwise, exit false.
- 110724 2 arguments: If \$1 is '!', exit true if \$2 is null, false if \$2 is not null.
- 110725 If \$1 is a unary primary, exit true if the unary test is true, false if the
110726 unary test is false.

- 110727 Otherwise, produce unspecified results.
- 110728 3 arguments: If \$2 is a binary primary, perform the binary test of \$1 and \$3.
- 110729 If \$1 is '!', negate the two-argument test of \$2 and \$3.
- 110730 OB XSI If \$1 is '(' and \$3 is ')', perform the unary test of \$2. On systems that do not support the XSI option, the results are unspecified if \$1 is '(' and \$3 is ')'.
110731
110732
- 110733 Otherwise, produce unspecified results.
- 110734 4 arguments: If \$1 is '!', negate the three-argument test of \$2, \$3, and \$4.
- 110735 OB XSI If \$1 is '(' and \$4 is ')', perform the two-argument test of \$2 and \$3.
110736 On systems that do not support the XSI option, the results are
110737 unspecified if \$1 is '(' and \$4 is ')'.
110738 Otherwise, the results are unspecified.
- 110739 >4 arguments: The results are unspecified.
- 110740 OB XSI On XSI-conformant systems, combinations of primaries and operators shall be
110741 evaluated using the precedence and associativity rules described previously.
110742 In addition, the string comparison binary primaries '=' and '!=' shall have
110743 a higher precedence than any unary primary.
- 110744 **STDIN**
- 110745 Not used.
- 110746 **INPUT FILES**
- 110747 None.
- 110748 **ENVIRONMENT VARIABLES**
- 110749 The following environment variables shall affect the execution of *test*:
- 110750 *LANG* Provide a default value for the internationalization variables that are unset or null.
110751 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
110752 variables used to determine the values of locale categories.)
- 110753 *LC_ALL* If set to a non-empty string value, override the values of all the other
110754 internationalization variables.
- 110755 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
110756 characters (for example, single-byte as opposed to multi-byte characters in
110757 arguments).
- 110758 *LC_MESSAGES*
- 110759 Determine the locale that should be used to affect the format and contents of
110760 diagnostic messages written to standard error.
- 110761 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 110762 **ASYNCHRONOUS EVENTS**
- 110763 Default.
- 110764 **STDOUT**
- 110765 Not used.

110766 **STDERR**

110767 The standard error shall be used only for diagnostic messages.

110768 **OUTPUT FILES**

110769 None.

110770 **EXTENDED DESCRIPTION**

110771 None.

110772 **EXIT STATUS**

110773 The following exit values shall be returned:

110774 0 *expression* evaluated to true.

110775 1 *expression* evaluated to false or *expression* was missing.

110776 >1 An error occurred.

110777 **CONSEQUENCES OF ERRORS**

110778 Default.

110779 **APPLICATION USAGE**

110780 The XSI extensions specifying the `-a` and `-o` binary primaries and the `'('` and `')'` operators
 110781 have been marked obsolescent. (Many expressions using them are ambiguously defined by the
 110782 grammar depending on the specific expressions being evaluated.) Scripts using these
 110783 expressions should be converted to the forms given below. Even though many implementations
 110784 will continue to support these obsolescent forms, scripts should be extremely careful when
 110785 dealing with user-supplied input that could be confused with these and other primaries and
 110786 operators. Unless the application developer knows all the cases that produce input to the script,
 110787 invocations like:

110788 `test "$1" -a "$2"`

110789 should be written as:

110790 `test "$1" && test "$2"`

110791 to avoid problems if a user supplied values such as \$1 set to `'!'` and \$2 set to the null string.
 110792 That is, in cases where maximal portability is of concern, replace:

110793 `test expr1 -a expr2`

110794 with:

110795 `test expr1 && test expr2`

110796 and replace:

110797 `test expr1 -o expr2`

110798 with:

110799 `test expr1 || test expr2`

110800 but note that, in *test*, `-a` has higher precedence than `-o` while `"&&"` and `"||"` have equal
 110801 precedence in the shell.

110802 Parentheses or braces can be used in the shell command language to effect grouping.

110803 Parentheses must be escaped when using *sh*; for example:

110804 `test \(expr1 -a expr2 \) -o expr3`

110805 This command is not always portable even on XSI-conformant systems depending on the

110806 expressions specified by *expr1*, *expr2*, and *expr3*. The following form can be used instead:

```
110807 ( test expr1 && test expr2 ) || test expr3
```

110808 The two commands:

```
110809 test "$1"
110810 test ! "$1"
```

110811 could not be used reliably on some historical systems. Unexpected results would occur if such a *string* expression were used and *\$1* expanded to '!', '(', or a known unary primary. Better constructs are:

```
110814 test -n "$1"
110815 test -z "$1"
```

110816 respectively.

110817 Historical systems have also been unreliable given the common construct:

```
110818 test "$response" = "expected string"
```

110819 One of the following is a more reliable form:

```
110820 test "X$response" = "Xexpected string"
110821 test "expected string" = "$response"
```

110822 Note that the second form assumes that *expected string* could not be confused with any unary primary. If *expected string* starts with '-', '(', '!', or even '=', the first form should be used instead. Using the preceding rules without the XSI marked extensions, any of the three comparison forms is reliable, given any input. (However, note that the strings are quoted in all cases.)

110827 Because the string comparison binary primaries, '=' and '!=', have a higher precedence than any unary primary in the greater than 4 argument case, unexpected results can occur if arguments are not properly prepared. For example, in:

```
110830 test -d $1 -o -d $2
```

110831 If *\$1* evaluates to a possible directory name of '=', the first three arguments are considered a string comparison, which shall cause a syntax error when the second *-d* is encountered. One of the following forms prevents this; the second is preferred:

```
110834 test \( -d "$1" \) -o \( -d "$2" \)
110835 test -d "$1" || test -d "$2"
```

110836 Also in the greater than 4 argument case:

```
110837 test "$1" = "bat" -a "$2" = "ball"
```

110838 syntax errors occur if *\$1* evaluates to '(' or '!'. One of the following forms prevents this; the third is preferred:

```
110840 test "X$1" = "Xbat" -a "X$2" = "Xball"
110841 test "$1" = "bat" && test "$2" = "ball"
110842 test "X$1" = "Xbat" && test "X$2" = "Xball"
```

110843 Note that none of the following examples are permitted by the syntax described:

```
110844 [-f file]
110845 [-f file ]
110846 [ -f file]
```

```
110847     [ -f file
110848     test -f file ]
```

110849 In the first two cases, if a utility named *[-f]* exists, that utility would be invoked, and not *test*. In
110850 the remaining cases, the brackets are mismatched, and the behavior is unspecified. However:

```
110851     test ! ]
```

110852 does have a defined meaning, and must exit with status 1. Similarly:

```
110853     test ]
```

110854 must exit with status 0.

110855 EXAMPLES

110856 1. Exit if there are not two or three arguments (two variations):

```
110857     if [ $# -ne 2 ] && [ $# -ne 3 ]; then exit 1; fi
110858     if [ $# -lt 2 ] || [ $# -gt 3 ]; then exit 1; fi
```

110859 2. Perform a *mkdir* if a directory does not exist:

```
110860     test ! -d tempdir && mkdir tempdir
```

110861 3. Wait for a file to become non-readable:

```
110862     while test -r thefile
110863     do
110864         sleep 30
110865     done
110866     echo "thefile" is no longer readable'
```

110867 4. Perform a command if the argument is one of three strings (two variations):

```
110868     if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
110869     then
110870         command
110871     fi
110872     case "$1" in
110873         pear|grape|apple) command ;;
110874     esac
```

110875 RATIONALE

110876 The KornShell-derived conditional command (double bracket `[[]]`) was removed from the shell
110877 command language description in an early proposal. Objections were raised that the real
110878 problem is misuse of the *test* command (`(I)`), and putting it into the shell is the wrong way to fix
110879 the problem. Instead, proper documentation and a new shell reserved word (`!`) are sufficient.

110880 Tests that require multiple *test* operations can be done at the shell level using individual
110881 invocations of the *test* command and shell logicals, rather than using the error-prone `-o` flag of
110882 *test*.

110883 XSI-conformant systems support more than four arguments.

110884 XSI-conformant systems support the combining of primaries with the following constructs:

110885 *expression1* **-a** *expression2*

110886 True if both *expression1* and *expression2* are true.

110887 *expression1 -o expression2*
 110888 True if at least one of *expression1* and *expression2* are true.

110889 (*expression*)
 110890 True if *expression* is true.

110891 In evaluating these more complex combined expressions, the following precedence rules are
 110892 used:

110893 The unary primaries have higher precedence than the algebraic binary primaries.

110894 The unary primaries have lower precedence than the string binary primaries.

110895 The unary and binary primaries have higher precedence than the unary *string* primary.

110896 The ! operator has higher precedence than the **-a** operator, and the **-a** operator has higher
 110897 precedence than the **-o** operator.

110898 The **-a** and **-o** operators are left associative.

110899 The parentheses can be used to alter the normal precedence and associativity.

110900 The BSD and System V versions of **-f** are not the same. The BSD definition was:

110901 **-f file** True if *file* exists and is not a directory.

110902 The SVID version (true if the file exists and is a regular file) was chosen for this volume of
 110903 POSIX.1-2017 because its use is consistent with the **-b**, **-c**, **-d**, and **-p** operands (*file* exists and is
 110904 a specific file type).

110905 The **-e** primary, possessing similar functionality to that provided by the C shell, was added
 110906 because it provides the only way for a shell script to find out if a file exists without trying to
 110907 open the file. Since implementations are allowed to add additional file types, a portable script
 110908 cannot use:

110909 `test -b foo -o -c foo -o -d foo -o -f foo -o -p foo`

110910 to find out if **foo** is an existing file. On historical BSD systems, the existence of a file could be
 110911 determined by:

110912 `test -f foo -o -d foo`

110913 but there was no easy way to determine that an existing file was a regular file. An early proposal
 110914 used the KornShell **-a** primary (with the same meaning), but this was changed to **-e** because
 110915 there were concerns about the high probability of humans confusing the **-a** primary with the **-a**
 110916 binary operator.

110917 The following options were not included in this volume of POSIX.1-2017, although they are
 110918 provided by some implementations. These operands should not be used by new
 110919 implementations for other purposes:

110920 **-k file** True if *file* exists and its sticky bit is set.

110921 **-C file** True if *file* is a contiguous file.

110922 **-V file** True if *file* is a version file.

110923 The following option was not included because it was undocumented in most implementations,
 110924 has been removed from some implementations (including System V), and the functionality is
 110925 provided by the shell (see [Section 2.6.2](#) (on page 2354)).

- 110926 **-l string** The length of the string *string*.
- 110927 The **-b**, **-c**, **-g**, **-p**, **-u**, and **-x** operands are derived from the SVID; historical BSD does not
110928 provide them. The **-k** operand is derived from System V; historical BSD does not provide it.
- 110929 On historical BSD systems, *test -w directory* always returned false because *test* tried to open the
110930 directory for writing, which always fails.
- 110931 Some additional primaries newly invented or from the KornShell appeared in an early proposal
110932 as part of the conditional command (`[[]]`): *s1 > s2*, *s1 < s2*, *str = pattern*, *str != pattern*, *f1 -nt f2*, *f1*
110933 *-ot f2*, and *f1 -ef f2*. They were not carried forward into the *test* utility when the conditional
110934 command was removed from the shell because they have not been included in the *test* utility
110935 built into historical implementations of the *sh* utility.
- 110936 The **-t file_descriptor** primary is shown with a mandatory argument because the grammar is
110937 ambiguous if it can be omitted. Historical implementations have allowed it to be omitted,
110938 providing a default of 1.
- 110939 It is noted that ' [' is not part of the portable filename character set; however, since it is required
110940 to be encoded by a single byte, and is part of the portable character set, the name of this utility
110941 forms a character string across all supported locales.
- 110942 **FUTURE DIRECTIONS**
- 110943 None.
- 110944 **SEE ALSO**
- 110945 [Section 1.1.1.4](#) (on page 2328), *find*
- 110946 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 110947 **CHANGE HISTORY**
- 110948 First released in Issue 2.
- 110949 **Issue 5**
- 110950 The FUTURE DIRECTIONS section is added.
- 110951 **Issue 6**
- 110952 The **-h** operand is added for symbolic links, and access permission requirements are clarified for
110953 the **-r**, **-w**, and **-x** operands to align with the IEEE P1003.2b draft standard.
- 110954 The normative text is reworded to avoid use of the term “must” for application requirements.
- 110955 The **-L** and **-S** operands are added for symbolic links and sockets.
- 110956 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/38 is applied, adding XSI margin
110957 marking and shading to a line in the OPERANDS section referring to the use of parentheses as
110958 arguments to the *test* utility.
- 110959 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/30 is applied, rewording the existence
110960 primaries for the *test* utility.
- 110961 **Issue 7**
- 110962 Austin Group Interpretation 1003.1-2001 #107 is applied.
- 110963 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0143 [291] is applied.
- 110964 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0191 [898], XCU/TC2-2008/0192
110965 [730], and XCU/TC2-2008/0193 [898] are applied.

110966 **NAME**

110967 time ‡'time a simple command

110968 **SYNOPSIS**110969 time [-p] *utility* [*argument...*]110970 **DESCRIPTION**

110971 The *time* utility shall invoke the utility named by the *utility* operand with arguments supplied as
 110972 the *argument* operands and write a message to standard error that lists timing statistics for the
 110973 utility. The message shall include the following information:

110974 The elapsed (real) time between invocation of *utility* and its termination.

110975 The User CPU time, equivalent to the sum of the *tms_utime* and *tms_cutime* fields returned
 110976 by the *times()* function defined in the System Interfaces volume of POSIX.1-2017 for the
 110977 process in which *utility* is executed.

110978 The System CPU time, equivalent to the sum of the *tms_stime* and *tms_cstime* fields
 110979 returned by the *times()* function for the process in which *utility* is executed.

110980 The precision of the timing shall be no less than the granularity defined for the size of the clock
 110981 tick unit on the system, but the results shall be reported in terms of standard time units (for
 110982 example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

110983 When *time* is used as part of a pipeline, the times reported are unspecified, except when it is the
 110984 sole command within a grouping command (see [Section 2.9.4.1](#), on page 2371) in that pipeline.
 110985 For example, the commands on the left are unspecified; those on the right report on utilities **a**
 110986 and **c**, respectively:

```
110987       time a | b | c       { time a; } | b | c
110988       a | b | time c       a | b | (time c)
```

110989 **OPTIONS**

110990 The *time* utility shall conform to XBD [Section 12.2](#) (on page 216).

110991 The following option shall be supported:

110992 **-p** Write the timing output to standard error in the format shown in the STDERR
 110993 section.

110994 **OPERANDS**

110995 The following operands shall be supported:

110996 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 110997 special built-in utilities in [Section 2.14](#) (on page 2384), the results are undefined.

110998 *argument* Any string to be supplied as an argument when invoking the utility named by the
 110999 *utility* operand.

111000 **STDIN**

111001 Not used.

111002 **INPUT FILES**

111003 None.

111004 **ENVIRONMENT VARIABLES**

111005 The following environment variables shall affect the execution of *time*:

111006 **LANG** Provide a default value for the internationalization variables that are unset or null.
 111007 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 111008 variables used to determine the values of locale categories.)

- 111009 *LC_ALL* If set to a non-empty string value, override the values of all the other
111010 internationalization variables.
- 111011 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
111012 characters (for example, single-byte as opposed to multi-byte characters in
111013 arguments).
- 111014 *LC_MESSAGES*
111015 Determine the locale that should be used to affect the format and contents of
111016 diagnostic and informative messages written to standard error.
- 111017 *LC_NUMERIC*
111018 Determine the locale for numeric formatting.
- 111019 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 111020 *PATH* Determine the search path that shall be used to locate the utility to be invoked; see
111021 XBD Chapter 8 (on page 173).
- 111022 **ASYNCHRONOUS EVENTS**
111023 Default.
- 111024 **STDOUT**
111025 Not used.
- 111026 **STDERR**
111027 If the *utility* utility is invoked, the standard error shall be used to write the timing statistics and
111028 may be used to write a diagnostic message if the utility terminates abnormally; otherwise, the
111029 standard error shall be used to write diagnostic messages and may also be used to write the
111030 timing statistics.
- 111031 If **-p** is specified, the following format shall be used for the timing statistics in the POSIX locale:
111032 "real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,
111033 <system seconds>
- 111034 where each floating-point number shall be expressed in seconds. The precision used may be less
111035 than the default six digits of %f, but shall be sufficiently precise to accommodate the size of the
111036 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits
111037 shall follow the radix character). The number of digits following the radix character shall be no
111038 less than one, even if this always results in a trailing zero. The implementation may append
111039 white space and additional information following the format shown here. The implementation
111040 may also prepend a single empty line before the format shown here.
- 111041 **OUTPUT FILES**
111042 None.
- 111043 **EXTENDED DESCRIPTION**
111044 None.
- 111045 **EXIT STATUS**
111046 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,
111047 the *time* utility shall exit with one of the following values:
- 111048 1-125 An error occurred in the *time* utility.
- 111049 126 The utility specified by *utility* was found but could not be invoked.

111050 127 The utility specified by *utility* could not be found.

111051 CONSEQUENCES OF ERRORS

111052 Default.

111053 APPLICATION USAGE

111054 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
 111055 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
 111056 utility exited with an error indication”. The value 127 was chosen because it is not commonly
 111057 used for other meanings; most utilities use small values for “normal error conditions” and the
 111058 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
 111059 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
 111060 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
 111061 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
 111062 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
 111063 any other reason.

111064 EXAMPLES

111065 It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by
 111066 placing pipelines and command lists in a single file; this file can then be invoked as a utility, and
 111067 the *time* applies to everything in the file.

111068 Alternatively, the following command can be used to apply *time* to a complex command:

```
111069 time sh -c 'complex-command-line'
```

111070 RATIONALE

111071 When the *time* utility was originally proposed to be included in the ISO POSIX-2:1993 standard,
 111072 questions were raised about its suitability for inclusion on the grounds that it was not useful for
 111073 conforming applications, specifically:

111074 The underlying CPU definitions from the System Interfaces volume of POSIX.1-2017 are
 111075 vague, so the numeric output could not be compared accurately between systems or even
 111076 between invocations.

111077 The creation of portable benchmark programs was outside the scope this volume of
 111078 POSIX.1-2017.

111079 However, *time* does fit in the scope of user portability. Human judgement can be applied to the
 111080 analysis of the output, and it could be very useful in hands-on debugging of applications or in
 111081 providing subjective measures of system performance. Hence it has been included in this
 111082 volume of POSIX.1-2017.

111083 The default output format has been left unspecified because historical implementations differ
 111084 greatly in their style of depicting this numeric output. The *-p* option was invented to provide
 111085 scripts with a common means of obtaining this information.

111086 In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather
 111087 than just a simple command. The POSIX definition has been worded to allow this
 111088 implementation. Consideration was given to invalidating this approach because of the historical
 111089 model from the C shell and System V shell. However, since the System V *time* utility historically
 111090 has not produced accurate results in pipeline timing (because the constituent processes are not
 111091 all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to
 111092 break historical KornShell usage.

111093 The term *utility* is used, rather than *command*, to highlight the fact that shell compound
 111094 commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*
 111095 includes user application programs and shell scripts, not just the standard utilities.

111096 **FUTURE DIRECTIONS**

111097 None.

111098 **SEE ALSO**111099 [Chapter 2](#) (on page 2345), *sh*111100 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)111101 XSH *times()*111102 **CHANGE HISTORY**

111103 First released in Issue 2.

111104 **Issue 6**

111105 This utility is marked as part of the User Portability Utilities option.

111106 **Issue 7**111107 The *time* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

111109 SD5-XCU-ERN-115 is applied, updating the example in the DESCRIPTION.

111110 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0144 [266] is applied.

111111 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0194 [723] is applied.

111112 **NAME**

111113 touch ‡change file access and modification times

111114 **SYNOPSIS**111115 touch [-acm] [-r *ref_file*|-t *time*|-d *date_time*] *file*...111116 **DESCRIPTION**111117 The *touch* utility shall change the last data modification timestamps, the last data access
111118 timestamps, or both.111119 The time used can be specified by the **-t** *time* option-argument, the corresponding *time* fields of
111120 the file referenced by the **-r** *ref_file* option-argument, or the **-d** *date_time* option-argument, as
111121 specified in the following sections. If none of these are specified, *touch* shall use the current time.111122 For each *file* operand, *touch* shall perform actions equivalent to the following functions defined
111123 in the System Interfaces volume of POSIX.1-2017:111124 1. If *file* does not exist:111125 a. The *creat()* function is called with the following arguments:111126 ‡*file* operand is used as the *path* argument.111127 ‡*mode* value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP,
111128 S_IWGRP, S_IROTH, and S_IWOTH is used as the *mode* argument.111129 b. The *futimens()* function is called with the following arguments:111130 ‡*fd* file descriptor opened in step 1a.111131 ‡*times* access time and the modification time, set as described in the OPTIONS
111132 section, are used as the first and second elements of the *times* array argument,
111133 respectively.111134 2. If *file* exists, the *utimensat()* function is called with the following arguments:111135 a. The AT_FDCWD special value is used as the *fd* argument.111136 b. The *file* operand is used as the *path* argument.111137 c. The access time and the modification time, set as described in the OPTIONS
111138 section, are used as the first and second elements of the *times* array argument,
111139 respectively.111140 d. The *flag* argument is set to zero.111141 **OPTIONS**111142 The *touch* utility shall conform to XBD [Section 12.2](#) (on page 216).

111143 The following options shall be supported:

111144 **-a** Change the access time of *file*. Do not change the modification time unless **-m** is
111145 also specified.111146 **-c** Do not create a specified *file* if it does not exist. Do not write any diagnostic
111147 messages concerning this condition.111148 **-d** *date_time* Use the specified *date_time* instead of the current time. The option-argument shall
111149 be a string of the form:111150 YYYY-MM-DDThh:mm:SS[.*frac*][*tz*]

111151 or:

111152 `YYYY-MM-DDThh:mm:SS[,frac][tz]`

111153 where:

111154 `YYYY` are at least four decimal digits giving the year.

111155 `MM`, `DD`, `hh`, `mm`, and `SS` are as with `-t time`.

111156 `T` is the time designator, and can be replaced by a single `<space>`.

111157 `[.frac]` and `[,frac]` are either empty, or a `<period>` (`'.'`) or a `<comma>`
 111158 (`'.'`) respectively, followed by one or more decimal digits, specifying a
 111159 fractional second.

111160 `[tz]` is either empty, signifying local time, or the letter `'Z'`, signifying UTC.
 111161 If `[tz]` is empty, the resulting time shall be affected by the value of the `TZ`
 111162 environment variable.

111163 If the resulting time precedes the Epoch, the behavior is implementation-defined. If
 111164 the time cannot be represented as the file's timestamp, `touch` shall exit immediately
 111165 with an error status.

111166 **-m** Change the modification time of `file`. Do not change the access time unless `-a` is
 111167 also specified.

111168 **-r ref_file** Use the corresponding time of the file named by the pathname `ref_file` instead of
 111169 the current time.

111170 **-t time** Use the specified `time` instead of the current time. The option-argument shall be a
 111171 decimal number of the form:

111172 `[[CC]YY]MMDDhhmm[.SS]`

111173 where each two digits represents the following:

111174 `MM` The month of the year [01,12].

111175 `DD` The day of the month [01,31].

111176 `hh` The hour of the day [00,23].

111177 `mm` The minute of the hour [00,59].

111178 `CC` The first two digits of the year (the century).

111179 `YY` The second two digits of the year.

111180 `SS` The second of the minute [00,60].

111181 Both `CC` and `YY` shall be optional. If neither is given, the current year shall be
 111182 assumed. If `YY` is specified, but `CC` is not, `CC` shall be derived as follows:

If YY is:	CC becomes:
[69,99]	19
[00,68]	20

111186 **Note:** It is expected that in a future version of this standard the default century inferred
 111187 from a 2-digit year will change. (This would apply to all commands accepting a
 111188 2-digit year as input.)

111189 The resulting time shall be affected by the value of the `TZ` environment variable. If
 111190 the resulting time value precedes the Epoch, the behavior is implementation-
 111191 defined. If the time is out of range for the file's timestamp, `touch` shall exit

111192 immediately with an error status. The range of valid times past the Epoch is
 111193 implementation-defined, but it shall extend to at least the time 0 hours, 0 minutes,
 111194 0 seconds, January 1, 2038, Coordinated Universal Time. Some implementations
 111195 may not be able to represent dates beyond January 18, 2038, because they use
 111196 **signed int** as a time holder.

111197 The range for *SS* is [00,60] rather than [00,59] because of leap seconds. If *SS* is 60,
 111198 and the resulting time, as affected by the *TZ* environment variable, does not refer
 111199 to a leap second, the resulting time shall be one second after a time where *SS* is 59.
 111200 If *SS* is not given a value, it is assumed to be zero.

111201 If neither the **-a** nor **-m** options were specified, *touch* shall behave as if both the **-a** and **-m**
 111202 options were specified.

111203 OPERANDS

111204 The following operands shall be supported:

111205 *file* A pathname of a file whose times shall be modified.

111206 STDIN

111207 Not used.

111208 INPUT FILES

111209 None.

111210 ENVIRONMENT VARIABLES

111211 The following environment variables shall affect the execution of *touch*:

111212 *LANG* Provide a default value for the internationalization variables that are unset or null.
 111213 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 111214 variables used to determine the values of locale categories.)

111215 *LC_ALL* If set to a non-empty string value, override the values of all the other
 111216 internationalization variables.

111217 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 111218 characters (for example, single-byte as opposed to multi-byte characters in
 111219 arguments).

111220 *LC_MESSAGES*

111221 Determine the locale that should be used to affect the format and contents of
 111222 diagnostic messages written to standard error.

111223 *XS* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

111224 *TZ* Determine the timezone to be used for interpreting the *time* option-argument. If *TZ*
 111225 is unset or null, an unspecified default timezone shall be used.

111226 ASYNCHRONOUS EVENTS

111227 Default.

111228 STDOUT

111229 Not used.

111230 STDERR

111231 The standard error shall be used only for diagnostic messages.

111232 **OUTPUT FILES**

111233 None.

111234 **EXTENDED DESCRIPTION**

111235 None.

111236 **EXIT STATUS**

111237 The following exit values shall be returned:

111238 0 The utility executed successfully and all requested changes were made.

111239 >0 An error occurred.

111240 **CONSEQUENCES OF ERRORS**

111241 Default.

111242 **APPLICATION USAGE**

111243 The interpretation of time is taken to be *seconds since the Epoch* (see XBD [Section 4.16](#), on page
 111244 113). It should be noted that implementations conforming to the System Interfaces volume of
 111245 POSIX.1-2017 do not take leap seconds into account when computing seconds since the Epoch.
 111246 When *SS=60* is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time
 111247 when *SS=59*.

111248 Although the *-t time* option-argument specifies values in 1969, the access time and modification
 111249 time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC).
 111250 Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid
 111251 hours in 1969 and there need not be any valid times in 1969.

111252 If the *T* time designator is replaced by a <space> for the *-d date_time* option-argument, the
 111253 <space> must be quoted to prevent the shell from splitting the argument.

111254 **EXAMPLES**

111255 Create or update a file called **dwc**; the resulting file has both the last data modification and last
 111256 data access timestamps set to November 12, 2007 at 10:15:30 local time:

111257 `touch -d 2007-11-12T10:15:30 dwc`

111258 Create or update a file called **nick**; the resulting file has both the last data modification and last
 111259 data access timestamps set to November 12, 2007 at 10:15:30 UTC:

111260 `touch -d 2007-11-12T10:15:30Z nick`

111261 Create or update a file called **gwc**; the resulting file has both the last data modification and last
 111262 data access timestamps set to November 12, 2007 at 10:15:30 local time with a fractional second
 111263 timestamp of .002 seconds:

111264 `touch -d 2007-11-12T10:15:30.002 gwc`

111265 Create or update a file called **ajosey**; the resulting file has both the last data modification and
 111266 last data access timestamps set to November 12, 2007 at 10:15:30 UTC with a fractional second
 111267 timestamp of .002 seconds:

111268 `touch -d "2007-11-12 10:15:30.002Z" ajosey`

111269 Create or update a file called **cathy**; the resulting file has both the last data modification and last
 111270 data access timestamps set to November 12, 2007 at 10:15:00 local time:

111271 `touch -t 200711121015 cathy`

111272 Create or update a file called **drepper**; the resulting file has both the last data modification and
 111273 last data access timestamps set to November 12, 2007 at 10:15:30 local time:

111274 touch -t 200711121015.30 drepper

111275 Create or update a file called **ebb9**; the resulting file has both the last data modification and last
111276 data access timestamps set to November 12, 2007 at 10:15:30 local time:

111277 touch -t 0711121015.30 ebb9

111278 Create or update a file called **eggert**; the resulting file has the last data access timestamp set to
111279 the corresponding time of the file named **mark** instead of the current time. If the file exists, the
111280 last data modification time is not changed:

111281 touch -a -r mark eggert

111282 RATIONALE

111283 The functionality of *touch* is described almost entirely through references to functions in the
111284 System Interfaces volume of POSIX.1-2017. In this way, there is no duplication of effort required
111285 for describing such side-effects as the relationship of user IDs to the user database, permissions,
111286 and so on.

111287 There are some significant differences between the *touch* utility in this volume of POSIX.1-2017
111288 and those in System V and BSD systems. They are upwards-compatible for historical
111289 applications from both implementations:

- 111290 1. In System V, an ambiguity exists when a pathname that is a decimal number leads the
111291 operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be
111292 *touched* to the current time. The **-t** *time* construct solves these problems for future
111293 conforming applications (note that the **-t** option is not historical practice).
- 111294 2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is
111295 treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates
111296 following the Epoch was included as recognition that some implementations are not able
111297 to represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

111298 The **-r** option was added because several comments requested this capability. This option was
111299 named **-f** in an early proposal, but was changed because the **-f** option is used in the BSD
111300 version of *touch* with a different meaning.

111301 At least one historical implementation of *touch* incremented the exit code if **-c** was specified and
111302 the file did not exist. This volume of POSIX.1-2017 requires exit status zero if no errors occur.

111303 In previous version of the standard, if at least two operands are specified, and the first operand
111304 is an eight or ten-digit decimal integer, the first operand was assumed to be a *date_time* operand.
111305 This usage was removed in this version of the standard since it had been marked obsolescent
111306 previously.

111307 The **-d** *date_time* format is an ISO 8601:2004 standard complete representation of date and time
111308 extended format with an optional decimal point or <comma> followed by a string of digits
111309 following the seconds portion to specify fractions of a second. It is not necessary to recognize
111310 "[+/-]hh:mm" and "[+/-]hh" to specify timezones other than local time and UTC. The *T*
111311 time designator in the ISO 8601:2004 standard extended format may be replaced by <space>.

111312 FUTURE DIRECTIONS

111313 None.

111314 SEE ALSO

111315 [date](#)

111316 XBD [Section 4.16](#) (on page 113), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216), [<sys/stat.h>](#)

111317 XSH [creat\(\)](#), [futimens\(\)](#), [time\(\)](#), [utime\(\)](#)

111318 CHANGE HISTORY

111319 First released in Issue 2.

111320 Issue 6

111321 The obsolescent *date_time* operand is removed.

111322 The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. This is a new requirement on POSIX implementations.

111325 The range for seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899:1999 standard, and to allow for positive leap seconds.

111327 Issue 7

111328 Austin Group Interpretation 1003.1-2001 #118 is applied.

111329 Austin Group Interpretation 1003.1-2001 #193 is applied, adding support for subsecond timestamps.

111331 SD5-XCU-ERN-45 is applied, adding a new paragraph to the RATIONALE.

111332 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

111333 SD5-XCU-ERN-110 is applied, updating the OPTIONS section.

111334 Changes are made related to support for finegrained timestamps.

111335 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0195 [474] is applied.

111336 **NAME**

111337 tput ‡change terminal characteristics

111338 **SYNOPSIS**111339 tput [-T *type*] *operand...*111340 **DESCRIPTION**

111341 The *tput* utility shall display terminal-dependent information. The manner in which this
 111342 information is retrieved is unspecified. The information displayed shall clear the terminal
 111343 screen, initialize the user's terminal, or reset the user's terminal, depending on the operand
 111344 given. The exact consequences of displaying this information are unspecified.

111345 **OPTIONS**111346 The *tput* utility shall conform to XBD [Section 12.2](#) (on page 216).

111347 The following option shall be supported:

111348 -T *type* Indicate the type of terminal. If this option is not supplied and the *TERM* variable
 111349 is unset or null, an unspecified default terminal type shall be used. The setting of
 111350 *type* shall take precedence over the value in *TERM*.

111351 **OPERANDS**

111352 The following strings shall be supported as operands by the implementation in the POSIX locale:

111353 **clear** Display the clear-screen sequence.

111354 **init** Display the sequence that initializes the user's terminal in an implementation-
 111355 defined manner.

111356 **reset** Display the sequence that resets the user's terminal in an implementation-defined
 111357 manner.

111358 If a terminal does not support any of the operations described by these operands, this shall not
 111359 be considered an error condition.

111360 **STDIN**

111361 Not used.

111362 **INPUT FILES**

111363 None.

111364 **ENVIRONMENT VARIABLES**111365 The following environment variables shall affect the execution of *tput*:

111366 LANG Provide a default value for the internationalization variables that are unset or null.
 111367 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 111368 variables used to determine the values of locale categories.)

111369 LC_ALL If set to a non-empty string value, override the values of all the other
 111370 internationalization variables.

111371 LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
 111372 characters (for example, single-byte as opposed to multi-byte characters in
 111373 arguments).

111374 LC_MESSAGES

111375 Determine the locale that should be used to affect the format and contents of
 111376 diagnostic messages written to standard error.

111377 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

111378 **TERM** Determine the terminal type. If this variable is unset or null, and if the **-T** option is
111379 not specified, an unspecified default terminal type shall be used.

111380 **ASYNCHRONOUS EVENTS**

111381 Default.

111382 **STDOUT**

111383 If standard output is a terminal device, it may be used for writing the appropriate sequence to
111384 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,
111385 undefined results occur.

111386 **STDERR**

111387 The standard error shall be used only for diagnostic messages.

111388 **OUTPUT FILES**

111389 None.

111390 **EXTENDED DESCRIPTION**

111391 None.

111392 **EXIT STATUS**

111393 The following exit values shall be returned:

111394 0 The requested string was written successfully.

111395 1 Unspecified.

111396 2 Usage error.

111397 3 No information is available about the specified terminal type.

111398 4 The specified operand is invalid.

111399 >4 An error occurred.

111400 **CONSEQUENCES OF ERRORS**

111401 If one of the operands is not available for the terminal, *tput* continues processing the remaining
111402 operands.

111403 **APPLICATION USAGE**

111404 The difference between resetting and initializing a terminal is left unspecified, as they vary
111405 greatly based on hardware types. In general, resetting is a more severe action.

111406 Some terminals use control characters to perform the stated functions, and on such terminals it
111407 might make sense to use *tput* to store the initialization strings in a file or environment variable
111408 for later use. However, because other terminals might rely on system calls to do this work, the
111409 standard output cannot be used in a portable manner, such as the following non-portable
111410 constructs:

```
111411 ClearVar=`tput clear`
111412 tput reset | mailx -s "Wake Up" ddg
```

111413 **EXAMPLES**

111414 1. Initialize the terminal according to the type of terminal in the environmental variable
111415 *TERM*. This command can be included in a **profile** file.

111416 `tput init`

111417 2. Reset a 450 terminal.
111418 tput -T 450 reset

111419 **RATIONALE**

111420 The list of operands was reduced to a minimum for the following reasons:

111421 The only features chosen were those that were likely to be used by human users interacting
111422 with a terminal.

111423 Specifying the full *terminfo* set was not considered desirable, but the standard developers
111424 did not want to select among operands.

111425 This volume of POSIX.1-2017 does not attempt to provide applications with sophisticated
111426 terminal handling capabilities, as that falls outside of its assigned scope and intersects with
111427 the responsibilities of other standards bodies.

111428 The difference between resetting and initializing a terminal is left unspecified as this varies
111429 greatly based on hardware types. In general, resetting is a more severe action.

111430 The exit status of 1 is historically reserved for finding out if a Boolean operand is not set.
111431 Although the operands were reduced to a minimum, the exit status of 1 should still be reserved
111432 for the Boolean operands, for those sites that wish to support them.

111433 **FUTURE DIRECTIONS**

111434 None.

111435 **SEE ALSO**

111436 [stty](#), [tabs](#)

111437 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

111438 **CHANGE HISTORY**

111439 First released in Issue 4.

111440 **Issue 6**

111441 This utility is marked as part of the User Portability Utilities option.

111442 **Issue 7**

111443 The *tput* utility is moved from the User Portability Utilities option to the Base. User Portability
111444 Utilities is now an option for interactive utilities.

111445 **NAME**

111446 tr ‡translate characters

111447 **SYNOPSIS**111448 tr [-c|-C] [-s] *string1 string2*111449 tr -s [-c|-C] *string1*111450 tr -d [-c|-C] *string1*111451 tr -ds [-c|-C] *string1 string2*111452 **DESCRIPTION**

111453 The *tr* utility shall copy the standard input to the standard output with substitution or deletion
 111454 of selected characters. The options specified and the *string1* and *string2* operands shall control
 111455 translations that occur while copying characters and single-character collating elements.

111456 **OPTIONS**111457 The *tr* utility shall conform to XBD [Section 12.2](#) (on page 216).

111458 The following options shall be supported:

111459 **-c** Complement the set of values specified by *string1*. See the EXTENDED
 111460 DESCRIPTION section.

111461 **-C** Complement the set of characters specified by *string1*. See the EXTENDED
 111462 DESCRIPTION section.

111463 **-d** Delete all occurrences of input characters that are specified by *string1*.

111464 **-s** Replace instances of repeated characters with a single character, as described in the
 111465 EXTENDED DESCRIPTION section.

111466 **OPERANDS**

111467 The following operands shall be supported:

111468 *string1, string2*

111469 Translation control strings. Each string shall represent a set of characters to be
 111470 converted into an array of characters used for the translation. For a detailed
 111471 description of how the strings are interpreted, see the EXTENDED DESCRIPTION
 111472 section.

111473 **STDIN**

111474 The standard input can be any type of file.

111475 **INPUT FILES**

111476 None.

111477 **ENVIRONMENT VARIABLES**111478 The following environment variables shall affect the execution of *tr*:

111479 **LANG** Provide a default value for the internationalization variables that are unset or null.
 111480 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 111481 variables used to determine the values of locale categories.)

111482 **LC_ALL** If set to a non-empty string value, override the values of all the other
 111483 internationalization variables.

111484 **LC_COLLATE**

111485 Determine the locale for the behavior of range expressions and equivalence classes.

111486	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes.
111487		
111488		
111489	<i>LC_MESSAGES</i>	
111490		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
111491		
111492	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
111493	ASYNCHRONOUS EVENTS	
111494		Default.
111495	STDOUT	
111496		The <i>tr</i> output shall be identical to the input, with the exception of the specified transformations.
111497	STDERR	
111498		The standard error shall be used only for diagnostic messages.
111499	OUTPUT FILES	
111500		None.
111501	EXTENDED DESCRIPTION	
111502		The operands <i>string1</i> and <i>string2</i> (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the constructs result in multi-character collating elements, <i>tr</i> shall exclude, without a diagnostic, those multi-character elements from the resulting array.
111503		
111504		
111505		
111506	<i>character</i>	Any character not described by one of the conventions below shall represent itself.
111507	<i>\octal</i>	Octal sequences can be used to represent characters with specific coded values. An octal sequence shall consist of a <backslash> followed by the longest sequence of one, two, or three-octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three-digit octal integer to be placed into the array. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <backslash> for each byte.
111508		
111509		
111510		
111511		
111512		
111513	<i>\character</i>	The <backslash>-escape sequences in XBD Table 5-1 (on page 121) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v') shall be supported. The results of using any other character, other than an octal digit, following the <backslash> are unspecified. Also, if there is no character following the <backslash>, the results are unspecified.
111514		
111515		
111516		
111517		
111518	<i>c-c</i>	In the POSIX locale, this construct shall represent the range of collating elements between the range endpoints (as long as neither endpoint is an octal sequence of the form <i>\octal</i>), inclusive, as defined by the collation sequence. The characters or collating elements in the range shall be placed in the array in ascending collation sequence. If the second endpoint precedes the starting endpoint in the collation sequence, it is unspecified whether the range of collating elements is empty, or this construct is treated as invalid. In locales other than the POSIX locale, this construct has unspecified behavior.
111519		
111520		
111521		
111522		
111523		
111524		
111525		
111526		If either or both of the range endpoints are octal sequences of the form <i>\octal</i> , this shall represent the range of specific coded values between the two range endpoints, inclusive.
111527		
111528		

111529	<code>[:class:]</code>	Represents all characters belonging to the defined character class, as defined by the current setting of the <code>LC_CTYPE</code> locale category. The following character class names shall be accepted when specified in <code>string1</code> :
111530		
111531		
111532		alnum blank digit lower punct upper
111533		alpha cntrl graph print space xdigit
111534	XSI	In addition, character class expressions of the form <code>[:name:]</code> shall be recognized in those locales where the <code>name</code> keyword has been given a charclass definition in the <code>LC_CTYPE</code> category.
111535		
111536		
111537		When both the <code>-d</code> and <code>-s</code> options are specified, any of the character class names shall be accepted in <code>string2</code> . Otherwise, only character class names lower or upper are valid in <code>string2</code> and then only if the corresponding character class (upper and lower , respectively) is specified in the same relative position in <code>string1</code> . Such a specification shall be interpreted as a request for case conversion. When <code>[:lower:]</code> appears in <code>string1</code> and <code>[:upper:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the toupper mapping in the <code>LC_CTYPE</code> category of the current locale. When <code>[:upper:]</code> appears in <code>string1</code> and <code>[:lower:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the tolower mapping in the <code>LC_CTYPE</code> category of the current locale. The first character from each mapping pair shall be in the array for <code>string1</code> and the second character from each mapping pair shall be in the array for <code>string2</code> in the same relative position.
111538		
111539		
111540		
111541		
111542		
111543		
111544		
111545		
111546		
111547		
111548		
111549		Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.
111550		
111551		If the name specified for <code>class</code> does not define a valid character class in the current locale, the behavior is undefined.
111552		
111553	<code>[=equiv=]</code>	Represents all characters or collating elements belonging to the same equivalence class as <code>equiv</code> , as defined by the current setting of the <code>LC_COLLATE</code> locale category. An equivalence class expression shall be allowed only in <code>string1</code> , or in <code>string2</code> when it is being used by the combined <code>-d</code> and <code>-s</code> options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.
111554		
111555		
111556		
111557		
111558	<code>[x*n]</code>	Represents <code>n</code> repeated occurrences of the character <code>x</code> . Because this expression is used to map multiple characters to one, it is only valid when it occurs in <code>string2</code> . If <code>n</code> is omitted or is zero, it shall be interpreted as large enough to extend the <code>string2</code> -based sequence to the length of the <code>string1</code> -based sequence. If <code>n</code> has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.
111559		
111560		
111561		
111562		
111563		
111564		When the <code>-d</code> option is not specified:
111565		If <code>string2</code> is present, each input character found in the array specified by <code>string1</code> shall be replaced by the character in the same relative position in the array specified by <code>string2</code> . If the array specified by <code>string2</code> is shorter than the one specified by <code>string1</code> , or if a character occurs more than once in <code>string1</code> , the results are unspecified.
111566		
111567		
111568		
111569		If the <code>-C</code> option is specified, the complements of the characters specified by <code>string1</code> (the set of all characters in the current character set, as defined by the current setting of <code>LC_CTYPE</code> , except for those actually specified in the <code>string1</code> operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of <code>LC_COLLATE</code> .
111570		
111571		
111572		

111573 If the `-c` option is specified, the complement of the values specified by *string1* shall be
111574 placed in the array in ascending order by binary value.

111575 Because the order in which characters specified by character class expressions or
111576 equivalence class expressions is undefined, such expressions should only be used if the
111577 intent is to map several characters into one. An exception is case conversion, as described
111578 previously.

111579 When the `-d` option is specified:

111580 Input characters found in the array specified by *string1* shall be deleted.

111581 When the `-C` option is specified with `-d`, all characters except those specified by *string1*
111582 shall be deleted. The contents of *string2* are ignored, unless the `-s` option is also specified.

111583 When the `-c` option is specified with `-d`, all values except those specified by *string1* shall
111584 be deleted. The contents of *string2* shall be ignored, unless the `-s` option is also specified.

111585 The same string cannot be used for both the `-d` and the `-s` option; when both options are
111586 specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be
111587 required.

111588 When the `-s` option is specified, after any deletions or translations have taken place, repeated
111589 sequences of the same character shall be replaced by one occurrence of the same character, if the
111590 character is found in the array specified by the last operand. If the last operand contains a
111591 character class, such as the following example:

```
111592 tr -s '[:space:]'
```

111593 the last operand's array shall contain all of the characters in that character class. However, in a
111594 case conversion, as described previously, such as:

```
111595 tr -s '[:upper:]' '[:lower:]'
```

111596 the last operand's array shall contain only those characters defined as the second characters in
111597 each of the **toupper** or **tolower** character pairs, as appropriate.

111598 An empty string used for *string1* or *string2* produces undefined results.

111599 EXIT STATUS

111600 The following exit values shall be returned:

111601 0 All input was processed successfully.

111602 >0 An error occurred.

111603 CONSEQUENCES OF ERRORS

111604 Default.

111605 APPLICATION USAGE

111606 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

111607 If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must
111608 use the full three digits to avoid ambiguity.

111609 When *string2* is shorter than *string1*, a difference results between historical System V and BSD
111610 systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible
111611 to do the following:

```
111612 tr 0123456789 d
```

111613 which would translate all digits to the letter 'd'. Since this area is specifically unspecified in

111614 this volume of POSIX.1-2017, both the BSD and System V behaviors are allowed, but a
 111615 conforming application cannot rely on the BSD behavior. It would have to code the example in
 111616 the following way:

```
111617 tr 0123456789 '[d*]'
```

111618 It should be noted that, despite similarities in appearance, the string operands used by *tr* are not
 111619 regular expressions.

111620 Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL
 111621 characters in its input stream. NUL characters can be stripped by using:

```
111622 tr -d '\000'
```

111623 EXAMPLES

111624 1. The following example creates a list of all words in **file1** one per line in **file2**, where a
 111625 word is taken to be a maximal string of letters.

```
111626 tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

111627 2. The next example translates all lowercase characters in **file1** to uppercase and writes the
 111628 results to standard output.

```
111629 tr "[:lower:]" "[:upper:]" <file1
```

111630 3. This example uses an equivalence class to identify accented variants of the base character
 111631 'e' in **file1**, which are stripped of diacritical marks and written to **file2**.

```
111632 tr "[=e]" "[e*]" <file1 >file2
```

111633 RATIONALE

111634 In some early proposals, an explicit option **-n** was added to disable the historical behavior of
 111635 stripping NUL characters from the input. It was considered that automatically stripping NUL
 111636 characters from the input was not correct functionality. However, the removal of **-n** in a later
 111637 proposal does not remove the requirement that *tr* correctly process NUL characters in its input
 111638 stream. NUL characters can be stripped by using *tr -d '\000'*.

111639 Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD
 111640 version has not needed the bracket characters for the repetition sequence. The *tr* utility syntax is
 111641 based more closely on the System V and XPG3 model while attempting to accommodate
 111642 historical BSD implementations. In the case of the short *string2* padding, the decision was to
 111643 unspecified the behavior and preserve System V and XPG3 scripts, which might find difficulty
 111644 with the BSD method. The assumption was made that BSD users of *tr* have to make
 111645 accommodations to meet the syntax defined here. Since it is possible to use the repetition
 111646 sequence to duplicate the desired behavior, whereas there is no simple way to achieve the
 111647 System V method, this was the correct, if not desirable, approach.

111648 The use of octal values to specify control characters, while having historical precedents, is not
 111649 portable. The introduction of escape sequences for control characters should provide the
 111650 necessary portability. It is recognized that this may cause some historical scripts to break.

111651 An early proposal included support for multi-character collating elements. It was pointed out
 111652 that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different;
 111653 ranges, for example, do not have a similar meaning (“any of the chars in the range matches”,
 111654 *versus* “translate each character in the range to the output counterpart”). As a result, the
 111655 previously included support for multi-character collating elements has been removed. What
 111656 remains are ranges in current collation order (to support, for example, accented characters),
 111657 character classes, and equivalence classes.

111658 In XPG3 the `[:class:]` and `[=equiv=]` conventions are shown with double brackets, as in RE syntax.
 111659 However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently,
 111660 `[:class:]` and `[=equiv=]` should be regarded as syntactical elements on a par with `[x*n]`, which is
 111661 not an RE bracket expression.

111662 The standard developers will consider changes to *tr* that allow it to translate characters between
 111663 different character encodings, or they will consider providing a new utility to accomplish this.

111664 On historical System V systems, a range expression requires enclosing square-brackets, such as:

```
111665 tr '[a-z]' '[A-Z]'
```

111666 However, BSD-based systems did not require the brackets, and this convention is used here to
 111667 avoid breaking large numbers of BSD scripts:

```
111668 tr a-z A-Z
```

111669 The preceding System V script will continue to work because the brackets, treated as regular
 111670 characters, are translated to themselves. However, any System V script that relied on "a-z"
 111671 representing the three characters 'a', '-', and 'z' have to be rewritten as "az-".

111672 The ISO POSIX-2:1993 standard had a `-c` option that behaved similarly to the `-C` option, but did
 111673 not supply functionality equivalent to the `-c` option specified in POSIX.1-2017.

111674 The earlier version also said that octal sequences referred to collating elements and could be
 111675 placed adjacent to each other to specify multi-byte characters. However, it was noted that this
 111676 caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were
 111677 intending to specify multi-byte characters or multiple single byte characters. POSIX.1-2017
 111678 specifies that octal sequences always refer to single byte binary values when used to specify an
 111679 endpoint of a range of collating elements.

111680 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
 111681 but this has been modified in this version.

111682 FUTURE DIRECTIONS

111683 None.

111684 SEE ALSO

111685 [sed](#)

111686 XBD [Table 5-1](#) (on page 121), [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

111687 CHANGE HISTORY

111688 First released in Issue 2.

111689 Issue 6

111690 The `-C` operand is added, and the description of the `-c` operand is changed to align with the
 111691 IEEE P1003.2b draft standard.

111692 The normative text is reworded to avoid use of the term "must" for application requirements.

111693 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/31 is applied, removing text describing
 111694 behavior on systems with bytes consisting of more than eight bits.

111695 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/32 is applied, updating an example in the
 111696 EXAMPLES section to avoid using unspecified behavior.

111697 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/33 is applied, making a correction to the
 111698 RATIONALE.

111699 **Issue 7**

- 111700 SD5-XCU-ERN-30 is applied.
- 111701 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 111702 Austin Group Interpretation 1003.1-2001 #132 is applied, adding rationale to the `\character`
111703 construct.
- 111704 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0145 [325] is applied.
- 111705 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0196 [663] is applied.

111706 NAME

111707 true — return true value

111708 SYNOPSIS

111709 true

111710 DESCRIPTION

111711 The *true* utility shall return with exit code zero.

111712 OPTIONS

111713 None.

111714 OPERANDS

111715 None.

111716 STDIN

111717 Not used.

111718 INPUT FILES

111719 None.

111720 ENVIRONMENT VARIABLES

111721 None.

111722 ASYNCHRONOUS EVENTS

111723 Default.

111724 STDOUT

111725 Not used.

111726 STDERR

111727 Not used.

111728 OUTPUT FILES

111729 None.

111730 EXTENDED DESCRIPTION

111731 None.

111732 EXIT STATUS

111733 Zero.

111734 CONSEQUENCES OF ERRORS

111735 None.

111736 APPLICATION USAGE

111737 This utility is typically used in shell scripts, as shown in the EXAMPLES section. The special
111738 built-in utility `:` is sometimes more efficient than *true*.

111739 EXAMPLES

111740 This command is executed forever:

```
111741 while true
111742 do
111743     command
111744 done
```

111745 **RATIONALE**

111746 The *true* utility has been retained in this volume of POSIX.1-2017, even though the shell special
111747 built-in `:` provides similar functionality, because *true* is widely used in historical scripts and is
111748 less cryptic to novice script readers.

111749 **FUTURE DIRECTIONS**

111750 None.

111751 **SEE ALSO**

111752 [Section 2.9](#) (on page 2365), *false*

111753 **CHANGE HISTORY**

111754 First released in Issue 2.

111755 **Issue 6**

111756 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/39 is applied, replacing the terms “None”
111757 and “Default” from the STDERR and EXIT STATUS sections, respectively, with terms as defined
111758 in [Section 1.4](#) (on page 2336).

111759 **NAME**

111760 tsort ‡topological sort

111761 **SYNOPSIS**111762 tsort [*file*]111763 **DESCRIPTION**111764 The *tsort* utility shall write to standard output a totally ordered list of items consistent with a
111765 partial ordering of items contained in the input.111766 The application shall ensure that the input consists of pairs of items (non-empty strings)
111767 separated by <blank> characters. Pairs of different items indicate ordering. Pairs of identical
111768 items indicate presence, but not ordering.111769 **OPTIONS**

111770 None.

111771 **OPERANDS**

111772 The following operand shall be supported:

111773 *file* A pathname of a text file to order. If no *file* operand is given, the standard input
111774 shall be used.111775 **STDIN**111776 The standard input shall be used if no *file* operand is specified, and shall be used if the *file*
111777 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
111778 the standard input shall not be used. See the INPUT FILES section.111779 **INPUT FILES**

111780 The input file shall be a text file.

111781 **ENVIRONMENT VARIABLES**111782 The following environment variables shall affect the execution of *tsort*:111783 *LANG* Provide a default value for the internationalization variables that are unset or null.
111784 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
111785 variables used to determine the values of locale categories.)111786 *LC_ALL* If set to a non-empty string value, override the values of all the other
111787 internationalization variables.111788 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
111789 characters (for example, single-byte as opposed to multi-byte characters in
111790 arguments and input files).111791 *LC_MESSAGES*111792 Determine the locale that should be used to affect the format and contents of
111793 diagnostic messages written to standard error.111794 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.111795 **ASYNCHRONOUS EVENTS**

111796 Default.

111797 **STDOUT**111798 The standard output shall be a text file consisting of the order list produced from the partially
111799 ordered input.

111800 STDERR

111801 The standard error shall be used only for diagnostic messages.

111802 OUTPUT FILES

111803 None.

111804 EXTENDED DESCRIPTION

111805 None.

111806 EXIT STATUS

111807 The following exit values shall be returned:

111808 0 Successful completion.

111809 >0 An error occurred.

111810 CONSEQUENCES OF ERRORS

111811 Default.

111812 APPLICATION USAGE

111813 The *LC_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not
111814 lexicographic, but depends on the pairs of items given as input.

111815 EXAMPLES

111816 The command:

```
111817 tsort <<EOF
111818 a b c c d e
111819 g g
111820 f g e f
111821 h h
111822 EOF
```

111823 produces the output:

```
111824 a
111825 b
111826 c
111827 d
111828 e
111829 f
111830 g
111831 h
```

111832 RATIONALE

111833 None.

111834 FUTURE DIRECTIONS

111835 None.

111836 SEE ALSO

111837 XBD [Chapter 8](#) (on page 173)

111838 CHANGE HISTORY

111839 First released in Issue 2.

111840 **Issue 6**

111841 The normative text is reworded to avoid use of the term “must” for application requirements.

111842 **Issue 7**

111843 Austin Group Interpretation 1003.1-2001 #092 is applied.

111844 The *tsort* utility is moved from the XSI option to the Base.

111845 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0146 [241] is applied.

111846 **NAME**

111847 tty — return user's terminal name

111848 **SYNOPSIS**

111849 tty

111850 **DESCRIPTION**

111851 The *tty* utility shall write to the standard output the name of the terminal that is open as
 111852 standard input. The name that is used shall be equivalent to the string that would be returned by
 111853 the *ttyname()* function defined in the System Interfaces volume of POSIX.1-2017.

111854 **OPTIONS**111855 The *tty* utility shall conform to XBD [Section 12.2](#) (on page 216).111856 **OPERANDS**

111857 None.

111858 **STDIN**

111859 While no input is read from standard input, standard input shall be examined to determine
 111860 whether or not it is a terminal, and, if so, to determine the name of the terminal.

111861 **INPUT FILES**

111862 None.

111863 **ENVIRONMENT VARIABLES**111864 The following environment variables shall affect the execution of *tty*:

111865 *LANG* Provide a default value for the internationalization variables that are unset or null.
 111866 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 111867 variables used to determine the values of locale categories.)

111868 *LC_ALL* If set to a non-empty string value, override the values of all the other
 111869 internationalization variables.

111870 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 111871 characters (for example, single-byte as opposed to multi-byte characters in
 111872 arguments).

111873 *LC_MESSAGES*

111874 Determine the locale that should be used to affect the format and contents of
 111875 diagnostic messages written to standard error and informative messages written to
 111876 standard output.

111877 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

111878 **ASYNCHRONOUS EVENTS**

111879 Default.

111880 **STDOUT**

111881 If standard input is a terminal device, a pathname of the terminal as specified by the *ttyname()*
 111882 function defined in the System Interfaces volume of POSIX.1-2017 shall be written in the
 111883 following format:

111884 "%s\n", <terminal name>

111885 Otherwise, a message shall be written indicating that standard input is not connected to a
 111886 terminal. In the POSIX locale, the *tty* utility shall use the format:

111887 "not a tty\n"

111888 **STDERR**

111889 The standard error shall be used only for diagnostic messages.

111890 **OUTPUT FILES**

111891 None.

111892 **EXTENDED DESCRIPTION**

111893 None.

111894 **EXIT STATUS**

111895 The following exit values shall be returned:

111896 0 Standard input is a terminal.

111897 1 Standard input is not a terminal.

111898 >1 An error occurred.

111899 **CONSEQUENCES OF ERRORS**

111900 Default.

111901 **APPLICATION USAGE**

111902 This utility checks the status of the file open as standard input against that of an
111903 implementation-defined set of files. It is possible that no match can be found, or that the match
111904 found need not be the same file as that which was opened for standard input (although they are
111905 the same device).

111906 **EXAMPLES**

111907 None.

111908 **RATIONALE**

111909 None.

111910 **FUTURE DIRECTIONS**

111911 None.

111912 **SEE ALSO**

111913 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

111914 XSH [isatty\(\)](#), [ttyname\(\)](#)

111915 **CHANGE HISTORY**

111916 First released in Issue 2.

111917 **Issue 5**

111918 The SYNOPSIS is changed to indicate two forms of the command, with the second form marked
111919 as obsolete. This is a clarification and does not change the functionality published in previous
111920 issues.

111921 **Issue 6**

111922 The obsolescent `-s` option is removed.

111923 **NAME**

111924 type ‡write a description of command type

111925 **SYNOPSIS**111926 XSI type *name*...111927 **DESCRIPTION**111928 The *type* utility shall indicate how each argument would be interpreted if used as a command
111929 name.111930 **OPTIONS**

111931 None.

111932 **OPERANDS**

111933 The following operand shall be supported:

111934 *name* A name to be interpreted.111935 **STDIN**

111936 Not used.

111937 **INPUT FILES**

111938 None.

111939 **ENVIRONMENT VARIABLES**111940 The following environment variables shall affect the execution of *type*:111941 *LANG* Provide a default value for the internationalization variables that are unset or null.
111942 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
111943 variables used to determine the values of locale categories.)111944 *LC_ALL* If set to a non-empty string value, override the values of all the other
111945 internationalization variables.111946 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
111947 characters (for example, single-byte as opposed to multi-byte characters in
111948 arguments).111949 *LC_MESSAGES*111950 Determine the locale that should be used to affect the format and contents of
111951 diagnostic messages written to standard error.111952 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.111953 *PATH* Determine the location of *name*, as described in XBD [Chapter 8](#) (on page 173).111954 **ASYNCHRONOUS EVENTS**

111955 Default.

111956 **STDOUT**111957 The standard output of *type* contains information about each operand in an unspecified format.111958 The information provided typically identifies the operand as a shell built-in, function, alias, or
111959 keyword, and where applicable, may display the operand's pathname.111960 **STDERR**

111961 The standard error shall be used only for diagnostic messages.

111962 **OUTPUT FILES**

111963 None.

111964 **EXTENDED DESCRIPTION**

111965 None.

111966 **EXIT STATUS**

111967 The following exit values shall be returned:

111968 0 Successful completion.

111969 >0 An error occurred.

111970 **CONSEQUENCES OF ERRORS**

111971 Default.

111972 **APPLICATION USAGE**

111973 Since *type* must be aware of the contents of the current shell execution environment (such as the
111974 lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell
111975 regular built-in. If it is called in a separate utility execution environment, such as one of the
111976 following:

111977 `nohup type writer`111978 `find . -type f | xargs type`

111979 it might not produce accurate results.

111980 **EXAMPLES**

111981 None.

111982 **RATIONALE**

111983 None.

111984 **FUTURE DIRECTIONS**

111985 None.

111986 **SEE ALSO**111987 *command*, *hash*111988 XBD [Chapter 8](#) (on page 173)111989 **CHANGE HISTORY**

111990 First released in Issue 2.

111991 **NAME**

111992 ulimit — set or report file size limit

111993 **SYNOPSIS**111994 XSI `ulimit [-f] [blocks]`111995 **DESCRIPTION**

111996 The *ulimit* utility shall set or report the file-size writing limit imposed on files written by the
 111997 shell and its child processes (files of any size may be read). Only a process with appropriate
 111998 privileges can increase the limit.

111999 **OPTIONS**112000 The *ulimit* utility shall conform to XBD [Section 12.2](#) (on page 216).

112001 The following option shall be supported:

112002 **-f** Set (or report, if no *blocks* operand is present), the file size limit in blocks. The **-f**
 112003 option shall also be the default case.

112004 **OPERANDS**

112005 The following operand shall be supported:

112006 *blocks* The number of 512-byte blocks to use as the new file size limit.112007 **STDIN**

112008 Not used.

112009 **INPUT FILES**

112010 None.

112011 **ENVIRONMENT VARIABLES**112012 The following environment variables shall affect the execution of *ulimit*:

112013 **LANG** Provide a default value for the internationalization variables that are unset or null.
 112014 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 112015 variables used to determine the values of locale categories.)

112016 **LC_ALL** If set to a non-empty string value, override the values of all the other
 112017 internationalization variables.

112018 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 112019 characters (for example, single-byte as opposed to multi-byte characters in
 112020 arguments).

112021 **LC_MESSAGES**

112022 Determine the locale that should be used to affect the format and contents of
 112023 diagnostic messages written to standard error.

112024 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

112025 **ASYNCHRONOUS EVENTS**

112026 Default.

112027 **STDOUT**

112028 The standard output shall be used when no *blocks* operand is present. If the current number of
 112029 blocks is limited, the number of blocks in the current limit shall be written in the following
 112030 format:

112031 "%d\n", <number of 512-byte blocks>

112032 If there is no current limit on the number of blocks, in the POSIX locale the following format

112033 shall be used:

112034 "unlimited\n"

112035 **STDERR**

112036 The standard error shall be used only for diagnostic messages.

112037 **OUTPUT FILES**

112038 None.

112039 **EXTENDED DESCRIPTION**

112040 None.

112041 **EXIT STATUS**

112042 The following exit values shall be returned:

112043 0 Successful completion.

112044 >0 A request for a higher limit was rejected or an error occurred.

112045 **CONSEQUENCES OF ERRORS**

112046 Default.

112047 **APPLICATION USAGE**

112048 Since *ulimit* affects the current shell execution environment, it is always provided as a shell regular built-in. If it is called in a separate utility execution environment, such as one of the following:

112049

112050

112051 nohup ulimit -f 10000

112052 env ulimit 10000

112053 it does not affect the file size limit of the caller's environment.

112054 Once a limit has been decreased by a process, it cannot be increased (unless appropriate privileges are involved), even back to the original system limit.

112055

112056 **EXAMPLES**

112057 Set the file size limit to 51 200 bytes:

112058 ulimit -f 100

112059 **RATIONALE**

112060 None.

112061 **FUTURE DIRECTIONS**

112062 None.

112063 **SEE ALSO**

112064 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

112065 XSH [ulimit\(\)](#)

112066 **CHANGE HISTORY**

112067 First released in Issue 2.

112068 **Issue 7**

112069 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112070 **NAME**

112071 umask — get or set the file mode creation mask

112072 **SYNOPSIS**112073 umask [-S] [*mask*]112074 **DESCRIPTION**

112075 The *umask* utility shall set the file mode creation mask of the current shell execution environment
 112076 (see [Section 2.12](#), on page 2381) to the value specified by the *mask* operand. This mask shall affect
 112077 the initial value of the file permission bits of subsequently created files. If *umask* is called in a
 112078 subshell or separate utility execution environment, such as one of the following:

```
112079 (umask 002)
112080 nohup umask ...
112081 find . -exec umask ... \;
```

112082 it shall not affect the file mode creation mask of the caller's environment.

112083 If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of
 112084 the file mode creation mask of the invoking process.

112085 **OPTIONS**112086 The *umask* utility shall conform to XBD [Section 12.2](#) (on page 216).

112087 The following option shall be supported:

112088 **-S** Produce symbolic output.

112089 The default output style is unspecified, but shall be recognized on a subsequent invocation of
 112090 *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

112091 **OPERANDS**

112092 The following operand shall be supported:

112093 *mask* A string specifying the new file mode creation mask. The string is treated in the
 112094 same way as the *mode* operand described in the EXTENDED DESCRIPTION
 112095 section for *chmod*.

112096 For a *symbolic_mode* value, the new value of the file mode creation mask shall be
 112097 the logical complement of the file permission bits portion of the file mode specified
 112098 by the *symbolic_mode* string.

112099 In a *symbolic_mode* value, the permissions *op* characters '+' and '-' shall be
 112100 interpreted relative to the current file mode creation mask; '+' shall cause the bits
 112101 for the indicated permissions to be cleared in the mask; '-' shall cause the bits for
 112102 the indicated permissions to be set in the mask.

112103 The interpretation of *mode* values that specify file mode bits other than the file
 112104 permission bits is unspecified.

112105 In the octal integer form of *mode*, the specified bits are set in the file mode creation
 112106 mask.

112107 The file mode creation mask shall be set to the resulting numeric value.

112108 The default output of a prior invocation of *umask* on the same system with no
 112109 operand also shall be recognized as a *mask* operand.

112110 **STDIN**

112111 Not used.

112112 **INPUT FILES**

112113 None.

112114 **ENVIRONMENT VARIABLES**112115 The following environment variables shall affect the execution of *umask*:

112116 *LANG* Provide a default value for the internationalization variables that are unset or null.
 112117 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 112118 variables used to determine the values of locale categories.)

112119 *LC_ALL* If set to a non-empty string value, override the values of all the other
 112120 internationalization variables.

112121 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 112122 characters (for example, single-byte as opposed to multi-byte characters in
 112123 arguments).

112124 *LC_MESSAGES*

112125 Determine the locale that should be used to affect the format and contents of
 112126 diagnostic messages written to standard error.

112127 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

112128 **ASYNCHRONOUS EVENTS**

112129 Default.

112130 **STDOUT**

112131 When the *mask* operand is not specified, the *umask* utility shall write a message to standard
 112132 output that can later be used as a *umask mask* operand.

112133 If *-S* is specified, the message shall be in the following format:

112134 "u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,
 112135 <other permissions>

112136 where the three values shall be combinations of letters from the set {*r, w, x*}; the presence of a
 112137 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

112138 If a *mask* operand is specified, there shall be no output written to standard output.

112139 **STDERR**

112140 The standard error shall be used only for diagnostic messages.

112141 **OUTPUT FILES**

112142 None.

112143 **EXTENDED DESCRIPTION**

112144 None.

112145 **EXIT STATUS**

112146 The following exit values shall be returned:

112147 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

112148 >0 An error occurred.

112149 CONSEQUENCES OF ERRORS

112150 Default.

112151 APPLICATION USAGE

112152 Since *umask* affects the current shell execution environment, it is generally provided as a shell
 112153 regular built-in.

112154 In contrast to the negative permission logic provided by the file mode creation mask and the
 112155 octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those
 112156 permissions that are left alone.

112157 EXAMPLES

112158 Either of the commands:

112159 `umask a=rx,ug+w`112160 `umask 002`

112161 sets the mode mask so that subsequently created files have their S_IWOTH bit cleared.

112162 After setting the mode mask with either of the above commands, the *umask* command can be
 112163 used to write out the current value of the mode mask:

112164 `$ umask`112165 `0002`

112166 (The output format is unspecified, but historical implementations use the octal integer mode
 112167 format.)

112168 `$ umask -S`112169 `u=rwx,g=rwx,o=rx`

112170 Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask*
 112171 utility.

112172 Assuming the mode mask is set as above, the command:

112173 `umask g-w`

112174 sets the mode mask so that subsequently created files have their S_IWGRP and S_IWOTH bits
 112175 cleared.

112176 The command:

112177 `umask -- -w`

112178 sets the mode mask so that subsequently created files have all their write bits cleared. Note that
 112179 *mask* operands `-r`, `-w`, `-x` or anything beginning with a <hyphen-minus>, must be preceded by
 112180 `--` to keep it from being interpreted as an option.

112181 RATIONALE

112182 Since *umask* affects the current shell execution environment, it is generally provided as a shell
 112183 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
 112184 of the following:

112185 `(umask 002)`112186 `nohup umask ...`112187 `find . -exec umask ... \;`

112188 it does not affect the file mode creation mask of the environment of the caller.

112189 The description of the historical utility was modified to allow it to use the symbolic modes of

112190 *chmod*. The `-s` option used in early proposals was changed to `-S` because `-s` could be confused
112191 with a *symbolic_mode* form of mask referring to the `S_ISUID` and `S_ISGID` bits.

112192 The default output style is unspecified to permit implementors to provide migration to the new
112193 symbolic style at the time most appropriate to their users. A `-o` flag to force octal mode output
112194 was omitted because the octal mode may not be sufficient to specify all of the information that
112195 may be present in the file mode creation mask when more secure file access permission checks
112196 are implemented.

112197 It has been suggested that trusted systems developers might appreciate ameliorating the
112198 requirement that the mode mask “affects” the file access permissions, since it seems access
112199 control lists might replace the mode mask to some degree. The wording has been changed to say
112200 that it affects the file permission bits, and it leaves the details of the behavior of how they affect
112201 the file access permissions to the description in the System Interfaces volume of POSIX.1-2017.

112202 **FUTURE DIRECTIONS**

112203 None.

112204 **SEE ALSO**

112205 [Chapter 2](#) (on page 2345), *chmod*

112206 [XBD Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

112207 XSH *umask()*

112208 **CHANGE HISTORY**

112209 First released in Issue 2.

112210 **Issue 6**

112211 The following new requirements on POSIX implementations derive from alignment with the
112212 Single UNIX Specification:

112213 The octal mode is supported.

112214 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/34 is applied, making a correction to the
112215 RATIONALE.

112216 **Issue 7**

112217 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112218 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0197 [584] is applied.

112219 **NAME**

112220 unalias — remove alias definitions

112221 **SYNOPSIS**112222 unalias *alias-name*...

112223 unalias -a

112224 **DESCRIPTION**

112225 The *unalias* utility shall remove the definition for each alias name specified. See [Section 2.3.1](#) (on
 112226 page 2348). The aliases shall be removed from the current shell execution environment; see
 112227 [Section 2.12](#) (on page 2381).

112228 **OPTIONS**112229 The *unalias* utility shall conform to XBD [Section 12.2](#) (on page 216).

112230 The following option shall be supported:

112231 **-a** Remove all alias definitions from the current shell execution environment.112232 **OPERANDS**

112233 The following operand shall be supported:

112234 *alias-name* The name of an alias to be removed.112235 **STDIN**

112236 Not used.

112237 **INPUT FILES**

112238 None.

112239 **ENVIRONMENT VARIABLES**112240 The following environment variables shall affect the execution of *unalias*:

112241 **LANG** Provide a default value for the internationalization variables that are unset or null.
 112242 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 112243 variables used to determine the values of locale categories.)

112244 **LC_ALL** If set to a non-empty string value, override the values of all the other
 112245 internationalization variables.

112246 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 112247 characters (for example, single-byte as opposed to multi-byte characters in
 112248 arguments).

112249 **LC_MESSAGES**

112250 Determine the locale that should be used to affect the format and contents of
 112251 diagnostic messages written to standard error.

112252 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.112253 **ASYNCHRONOUS EVENTS**

112254 Default.

112255 **STDOUT**

112256 Not used.

112257 **STDERR**

112258 The standard error shall be used only for diagnostic messages.

112259 OUTPUT FILES

112260 None.

112261 EXTENDED DESCRIPTION

112262 None.

112263 EXIT STATUS

112264 The following exit values shall be returned:

112265 0 Successful completion.

112266 >0 One of the *alias-name* operands specified did not represent a valid alias definition, or an
112267 error occurred.

112268 CONSEQUENCES OF ERRORS

112269 Default.

112270 APPLICATION USAGE

112271 Since *unalias* affects the current shell execution environment, it is generally provided as a shell
112272 regular built-in.

112273 EXAMPLES

112274 None.

112275 RATIONALE

112276 The *unalias* description is based on that from historical KornShell implementations. Known
112277 differences exist between that and the C shell. The KornShell version was adopted to be
112278 consistent with all the other KornShell features in this volume of POSIX.1-2017, such as
112279 command line editing.

112280 The *-a* option is the equivalent of the *unalias ** form of the C shell and is provided to address
112281 security concerns about unknown aliases entering the environment of a user (or application)
112282 through the allowable implementation-defined predefined alias route or as a result of an *ENV*
112283 file. (Although *unalias* could be used to simplify the “secure” shell script shown in the *command*
112284 rationale, it does not obviate the need to quote all command names. An initial call to *unalias -a*
112285 would have to be quoted in case there was an alias for *unalias*.)

112286 FUTURE DIRECTIONS

112287 None.

112288 SEE ALSO

112289 [Chapter 2](#) (on page 2345), *alias*

112290 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

112291 CHANGE HISTORY

112292 First released in Issue 4.

112293 Issue 6

112294 This utility is marked as part of the User Portability Utilities option.

112295 Issue 7

112296 The *unalias* utility is moved from the User Portability Utilities option to the Base. User
112297 Portability Utilities is now an option for interactive utilities.

112298 **NAME**

112299 uname — return system name

112300 **SYNOPSIS**

112301 uname [-amnrsv]

112302 **DESCRIPTION**

112303 By default, the *uname* utility shall write the operating system name to standard output. When
 112304 options are specified, symbols representing one or more system characteristics shall be written to
 112305 the standard output. The format and contents of the symbols are implementation-defined. On
 112306 systems conforming to the System Interfaces volume of POSIX.1-2017, the symbols written shall
 112307 be those supported by the *uname()* function as defined in the System Interfaces volume of
 112308 POSIX.1-2017.

112309 **OPTIONS**112310 The *uname* utility shall conform to XBD [Section 12.2](#) (on page 216).

112311 The following options shall be supported:

- 112312 **-a** Behave as though all of the options **-mnrsv** were specified.
- 112313 **-m** Write the name of the hardware type on which the system is running to standard
 112314 output.
- 112315 **-n** Write the name of this node within an implementation-defined communications
 112316 network.
- 112317 **-r** Write the current release level of the operating system implementation.
- 112318 **-s** Write the name of the implementation of the operating system.
- 112319 **-v** Write the current version level of this release of the operating system
 112320 implementation.

112321 If no options are specified, the *uname* utility shall write the operating system name, as if the **-s**
 112322 option had been specified.

112323 **OPERANDS**

112324 None.

112325 **STDIN**

112326 Not used.

112327 **INPUT FILES**

112328 None.

112329 **ENVIRONMENT VARIABLES**112330 The following environment variables shall affect the execution of *uname*:

- 112331 **LANG** Provide a default value for the internationalization variables that are unset or null.
 112332 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 112333 variables used to determine the values of locale categories.)
- 112334 **LC_ALL** If set to a non-empty string value, override the values of all the other
 112335 internationalization variables.
- 112336 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 112337 characters (for example, single-byte as opposed to multi-byte characters in
 112338 arguments).

112339 **LC_MESSAGES**
 112340 Determine the locale that should be used to affect the format and contents of
 112341 diagnostic messages written to standard error.

112342 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

112343 **ASYNCHRONOUS EVENTS**
 112344 Default.

112345 **STDOUT**
 112346 By default, the output shall be a single line of the following form:
 112347 "%s\n", <sysname>
 112348 If the **-a** option is specified, the output shall be a single line of the following form:
 112349 "%s %s %s %s %s\n", <sysname>, <nodename>, <release>,
 112350 <version>, <machine>
 112351 Additional implementation-defined symbols may be written; all such symbols shall be written at
 112352 the end of the line of output before the <newline>.
 112353 If options are specified to select different combinations of the symbols, only those symbols shall
 112354 be written, in the order shown above for the **-a** option. If a symbol is not selected for writing, its
 112355 corresponding trailing <blank> characters also shall not be written.

112356 **STDERR**
 112357 The standard error shall be used only for diagnostic messages.

112358 **OUTPUT FILES**
 112359 None.

112360 **EXTENDED DESCRIPTION**
 112361 None.

112362 **EXIT STATUS**
 112363 The following exit values shall be returned:
 112364 0 The requested information was successfully written.
 112365 >0 An error occurred.

112366 **CONSEQUENCES OF ERRORS**
 112367 Default.

112368 **APPLICATION USAGE**
 112369 Note that any of the symbols could include embedded <space> characters, which may affect
 112370 parsing algorithms if multiple options are selected for output.
 112371 The node name is typically a name that the system uses to identify itself for inter-system
 112372 communication addressing.

112373 **EXAMPLES**
 112374 The following command:
 112375 `uname -sr`
 112376 writes the operating system name and release level, separated by one or more <blank>
 112377 characters.

112378 **RATIONALE**

112379 It was suggested that this utility cannot be used portably since the format of the symbols is
112380 implementation-defined. The POSIX.1 working group could not achieve consensus on defining
112381 these formats in the underlying *uname()* function, and there was no expectation that this volume
112382 of POSIX.1-2017 would be any more successful. Some applications may still find this historical
112383 utility of value. For example, the symbols could be used for system log entries or for comparison
112384 with operator or user input.

112385 **FUTURE DIRECTIONS**

112386 None.

112387 **SEE ALSO**

112388 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

112389 XSH [uname\(\)](#)

112390 **CHANGE HISTORY**

112391 First released in Issue 2.

112392 **NAME**

112393 uncompress — expand compressed data

112394 **SYNOPSIS**112395 XSI uncompress [-cfv] [*file...*]112396 **DESCRIPTION**

112397 The *uncompress* utility shall restore files to their original state after they have been compressed
 112398 using the *compress* utility. If no files are specified, the standard input shall be uncompressed to
 112399 the standard output. If the invoking process has appropriate privileges, the ownership, modes,
 112400 access time, and modification time of the original file shall be preserved.

112401 This utility shall support the uncompressing of any files produced by the *compress* utility on the
 112402 same implementation. For files produced by *compress* on other systems, *uncompress* supports 9 to
 112403 14-bit compression (see *compress*, -b); it is implementation-defined whether values of -b greater
 112404 than 14 are supported.

112405 **OPTIONS**

112406 The *uncompress* utility shall conform to XBD Section 12.2 (on page 216), except that Guideline 1
 112407 does apply since the utility name has ten letters.

112408 The following options shall be supported:

- 112409 -c Write to standard output; no files are changed.
- 112410 -f Do not prompt for overwriting files. Except when run in the background, if -f is
 112411 not given the user shall be prompted as to whether an existing file should be
 112412 overwritten. If the standard input is not a terminal and -f is not given, *uncompress*
 112413 shall write a diagnostic message to standard error and exit with a status greater
 112414 than zero.
- 112415 -v Write messages to standard error concerning the expansion of each file.

112416 **OPERANDS**

112417 The following operand shall be supported:

- 112418 *file* A pathname of a file. If *file* already has the .Z suffix specified, it shall be used as the
 112419 input file and the output file shall be named **file** with the .Z suffix removed.
 112420 Otherwise, *file* shall be used as the name of the output file and **file** with the .Z
 112421 suffix appended shall be used as the input file.

112422 **STDIN**

112423 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.

112424 **INPUT FILES**

112425 Input files shall be in the format produced by the *compress* utility.

112426 **ENVIRONMENT VARIABLES**

112427 The following environment variables shall affect the execution of *uncompress*:

- 112428 *LANG* Provide a default value for the internationalization variables that are unset or null.
 112429 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 112430 variables used to determine the values of locale categories.)
- 112431 *LC_ALL* If set to a non-empty string value, override the values of all the other
 112432 internationalization variables.

- 112433 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
- 112434
112435
- 112436 *LC_MESSAGES*
112437 Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
112438
- 112439 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 112440 **ASYNCHRONOUS EVENTS**
- 112441 Default.
- 112442 **STDOUT**
112443 When there are no *file* operands or the *-c* option is specified, the uncompressed output is written to standard output.
112444
- 112445 **STDERR**
112446 Prompts shall be written to the standard error output under the conditions specified in the DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is otherwise unspecified. Otherwise, the standard error output shall be used only for diagnostic messages.
112447
112448
112449
- 112450 **OUTPUT FILES**
112451 Output files are the same as the respective input files to *compress*.
- 112452 **EXTENDED DESCRIPTION**
112453 None.
- 112454 **EXIT STATUS**
112455 The following exit values shall be returned:
112456 0 Successful completion.
112457 >0 An error occurred.
- 112458 **CONSEQUENCES OF ERRORS**
112459 The input file remains unmodified.
- 112460 **APPLICATION USAGE**
112461 The limit of 14 on the *compress -b bits* argument is to achieve portability to all systems (within the restrictions imposed by the lack of an explicit published file format). Some implementations based on 16-bit architectures cannot support 15 or 16-bit uncompression.
112462
112463
- 112464 **EXAMPLES**
112465 None.
- 112466 **RATIONALE**
112467 None.
- 112468 **FUTURE DIRECTIONS**
112469 None.
- 112470 **SEE ALSO**
112471 *compress*, *zcat*
112472 XBD Chapter 8 (on page 173), Section 12.2 (on page 216)

112473 **CHANGE HISTORY**

112474 First released in Issue 4.

112475 **Issue 6**

112476 The normative text is reworded to avoid use of the term “must” for application requirements.

112477 **Issue 7**

112478 SD5-XCU-ERN-26 is applied, clarifying that this utility is allowed to break the Utility Syntax Guidelines by having ten letters in its name.

112480 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112481 **NAME**112482 unexpand \ddagger convert spaces to tabs112483 **SYNOPSIS**112484 unexpand [-a|-t *tablist*] [*file...*]112485 **DESCRIPTION**

112486 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>
 112487 characters at the beginning of each line into the maximum number of <tab> characters followed
 112488 by the minimum number of <space> characters needed to fill the same column positions
 112489 originally filled by the translated <blank> characters. By default, tabstops shall be set at every
 112490 eighth column position. Each <backspace> shall be copied to the output, and shall cause the
 112491 column position count for tab calculations to be decremented; the count shall never be
 112492 decremented to a value less than one.

112493 **OPTIONS**112494 The *unexpand* utility shall conform to XBD [Section 12.2](#) (on page 216).

112495 The following options shall be supported:

112496 **-a** In addition to translating <blank> characters at the beginning of each line,
 112497 translate all sequences of two or more <blank> characters immediately preceding a
 112498 tab stop to the maximum number of <tab> characters followed by the minimum
 112499 number of <space> characters needed to fill the same column positions originally
 112500 filled by the translated <blank> characters.

112501 **-t *tablist*** Specify the tab stops. The application shall ensure that the *tablist* option-argument
 112502 is a single argument consisting of a single positive decimal integer or multiple
 112503 positive decimal integers, separated by <blank> or <comma> characters, in
 112504 ascending order. If a single number is given, tabs shall be set *tablist* column
 112505 positions apart instead of the default 8. If multiple numbers are given, the tabs
 112506 shall be set at those specific column positions.

112507 The application shall ensure that each tab-stop position *N* is an integer value
 112508 greater than zero, and the list shall be in strictly ascending order. This is taken to
 112509 mean that, from the start of a line of output, tabbing to position *N* shall cause the
 112510 next character output to be in the (*N*+1)th column position on that line. When the
 112511 **-t** option is not specified, the default shall be the equivalent of specifying **-t 8**
 112512 (except for the interaction with **-a**, described below).

112513 No <space>-to-<tab> conversions shall occur for characters at positions beyond
 112514 the last of those specified in a multiple tab-stop list.

112515 When **-t** is specified, the presence or absence of the **-a** option shall be ignored;
 112516 conversion shall not be limited to the processing of leading <blank> characters.

112517 **OPERANDS**

112518 The following operand shall be supported:

112519 *file* A pathname of a text file to be used as input.112520 **STDIN**

112521 See the INPUT FILES section.

112522 **INPUT FILES**

112523 The input files shall be text files.

112524 **ENVIRONMENT VARIABLES**

112525 The following environment variables shall affect the execution of *unexpand*:

112526 *LANG* Provide a default value for the internationalization variables that are unset or null.
 112527 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 112528 variables used to determine the values of locale categories.)

112529 *LC_ALL* If set to a non-empty string value, override the values of all the other
 112530 internationalization variables.

112531 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 112532 characters (for example, single-byte as opposed to multi-byte characters in
 112533 arguments and input files), the processing of <tab> and <space> characters, and
 112534 for the determination of the width in column positions each character would
 112535 occupy on an output device.

112536 *LC_MESSAGES*

112537 Determine the locale that should be used to affect the format and contents of
 112538 diagnostic messages written to standard error.

112539 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

112540 **ASYNCHRONOUS EVENTS**

112541 Default.

112542 **STDOUT**

112543 The standard output shall be equivalent to the input files with the specified <space>-to-<tab>
 112544 conversions.

112545 **STDERR**

112546 The standard error shall be used only for diagnostic messages.

112547 **OUTPUT FILES**

112548 None.

112549 **EXTENDED DESCRIPTION**

112550 None.

112551 **EXIT STATUS**

112552 The following exit values shall be returned:

112553 0 Successful completion.

112554 >0 An error occurred.

112555 **CONSEQUENCES OF ERRORS**

112556 Default.

112557 **APPLICATION USAGE**

112558 One non-intuitive aspect of *unexpand* is its restriction to leading <space> characters when neither
 112559 *-a* nor *-t* is specified. Users who always want to convert all <space> characters in a file can
 112560 easily alias *unexpand* to use the *-a* or *-t 8* option.

112561 **EXAMPLES**

112562 None.

112563 **RATIONALE**

112564 On several occasions, consideration was given to adding a *-t* option to the *unexpand* utility to
 112565 complement the *-t* in *expand* (see *expand*). The historical intent of *unexpand* was to translate
 112566 multiple <blank> characters into tab stops, where tab stops were a multiple of eight column

112567 positions on most UNIX systems. An early proposal omitted `-t` because it seemed outside the
112568 scope of the User Portability Utilities option; it was not described in any of the base documents
112569 for IEEE Std 1003.2-1992. However, hard-coding tab stops every eight columns was not suitable
112570 for the international community and broke historical precedents for some vendors in the
112571 FORTRAN community, so `-t` was restored in conjunction with the list of valid extension
112572 categories considered by the standard developers. Thus, *unexpand* is now the logical converse of
112573 *expand*.

112574 **FUTURE DIRECTIONS**

112575 None.

112576 **SEE ALSO**

112577 *expand*, *tabs*

112578 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

112579 **CHANGE HISTORY**

112580 First released in Issue 4.

112581 **Issue 6**

112582 This utility is marked as part of the User Portability Utilities option.

112583 The definition of the `LC_CTYPE` environment variable is changed to align with the
112584 IEEE P1003.2b draft standard.

112585 The normative text is reworded to avoid use of the term “must” for application requirements.

112586 **Issue 7**

112587 The *unexpand* utility is moved from the User Portability Utilities option to the Base. User
112588 Portability Utilities is now an option for interactive utilities.

112589 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112590 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0198 [885] is applied.

112591 **NAME**112592 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)112593 **SYNOPSIS**112594 XSI unget [-ns] [-r *SID*] *file...*112595 **DESCRIPTION**112596 The *unget* utility shall reverse the effect of a *get -e* done prior to creating the intended new delta.112597 **OPTIONS**112598 The *unget* utility shall conform to XBD [Section 12.2](#) (on page 216).

112599 The following options shall be supported:

112600 **-r** *SID* Uniquely identify which delta is no longer intended. (This would have been
 112601 specified by *get* as the new delta.) The use of this option is necessary only if two or
 112602 more outstanding *get* commands for editing on the same SCCS file were done by
 112603 the same person (login name).

112604 **-s** Suppress the writing to standard output of the intended delta's SID.

112605 **-n** Retain the file that was obtained by *get*, which would normally be removed from
 112606 the current directory.

112607 **OPERANDS**

112608 The following operands shall be supported:

112609 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget*
 112610 utility shall behave as though each file in the directory were specified as a named
 112611 file, except that non-SCCS files (last component of the pathname does not begin
 112612 with **s**.) and unreadable files shall be silently ignored.

112613 If exactly one *file* operand appears, and it is '-', the standard input shall be read;
 112614 each line of the standard input shall be taken to be the name of an SCCS file to be
 112615 processed. Non-SCCS files and unreadable files shall be silently ignored.

112616 **STDIN**

112617 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each
 112618 line of the text file shall be interpreted as an SCCS pathname.

112619 **INPUT FILES**

112620 Any SCCS files processed shall be files of an unspecified format.

112621 **ENVIRONMENT VARIABLES**112622 The following environment variables shall affect the execution of *unget*:

112623 **LANG** Provide a default value for the internationalization variables that are unset or null.
 112624 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 112625 variables used to determine the values of locale categories.)

112626 **LC_ALL** If set to a non-empty string value, override the values of all the other
 112627 internationalization variables.

112628 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 112629 characters (for example, single-byte as opposed to multi-byte characters in
 112630 arguments and input files).

112631 **LC_MESSAGES**

112632 Determine the locale that should be used to affect the format and contents of
 112633 diagnostic messages written to standard error.

- 112634 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 112635 **ASYNCHRONOUS EVENTS**
- 112636 Default.
- 112637 **STDOUT**
- 112638 The standard output shall consist of a line for each file, in the following format:
- 112639 "%s\n", <*SID removed from file*>
- 112640 If there is more than one named file or if a directory or standard input is named, each pathname
112641 shall be written before each of the preceding lines:
- 112642 "\n%s:\n", <*pathname*>
- 112643 **STDERR**
- 112644 The standard error shall be used only for diagnostic messages.
- 112645 **OUTPUT FILES**
- 112646 Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a
112647 locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and
112648 deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.
- 112649 **EXTENDED DESCRIPTION**
- 112650 None.
- 112651 **EXIT STATUS**
- 112652 The following exit values shall be returned:
- 112653 0 Successful completion.
- 112654 >0 An error occurred.
- 112655 **CONSEQUENCES OF ERRORS**
- 112656 Default.
- 112657 **APPLICATION USAGE**
- 112658 None.
- 112659 **EXAMPLES**
- 112660 None.
- 112661 **RATIONALE**
- 112662 None.
- 112663 **FUTURE DIRECTIONS**
- 112664 None.
- 112665 **SEE ALSO**
- 112666 *delta*, *get*, *sact*
- 112667 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 112668 **CHANGE HISTORY**
- 112669 First released in Issue 2.
- 112670 **Issue 6**
- 112671 The normative text is reworded to avoid use of the term “must” for application requirements.

112672 **Issue 7**
112673

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112674 **NAME**

112675 **uniq** — report or filter out repeated lines in a file

112676 **SYNOPSIS**

112677 **uniq** [-c|-d|-u] [-f *fields*] [-s *char*] [*input_file* [*output_file*]]

112678 **DESCRIPTION**

112679 The *uniq* utility shall read an input file comparing adjacent lines, and write one copy of each
 112680 input line on the output. The second and succeeding copies of repeated adjacent input lines shall
 112681 not be written. The trailing <newline> of each line in the input shall be ignored when doing
 112682 comparisons.

112683 Repeated lines in the input shall not be detected if they are not adjacent.

112684 **OPTIONS**

112685 The *uniq* utility shall conform to XBD [Section 12.2](#) (on page 216), except that '+' may be
 112686 recognized as an option delimiter as well as '-'.
 112687 The following options shall be supported:

112687 The following options shall be supported:

112688 **-c** Precede each output line with a count of the number of times the line occurred in
 112689 the input.

112690 **-d** Suppress the writing of lines that are not repeated in the input.

112691 **-f *fields*** Ignore the first *fields* fields on each input line when doing comparisons, where
 112692 *fields* is a positive decimal integer. A field is the maximal string matched by the
 112693 basic regular expression:

112694 [[[:blank:]]*^[[:blank:]]*]

112695 If the *fields* option-argument specifies more fields than appear on an input line, a
 112696 null string shall be used for comparison.

112697 **-s *chars*** Ignore the first *chars* characters when doing comparisons, where *chars* shall be a
 112698 positive decimal integer. If specified in conjunction with the **-f** option, the first
 112699 *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-
 112700 argument specifies more characters than remain on an input line, a null string shall
 112701 be used for comparison.

112702 **-u** Suppress the writing of lines that are repeated in the input.

112703 **OPERANDS**

112704 The following operands shall be supported:

112705 *input_file* A pathname of the input file. If the *input_file* operand is not specified, or if the
 112706 *input_file* is '-', the standard input shall be used.

112707 *output_file* A pathname of the output file. If the *output_file* operand is not specified, the
 112708 standard output shall be used. The results are unspecified if the file named by
 112709 *output_file* is the file named by *input_file*.

112710 **STDIN**

112711 The standard input shall be used only if no *input_file* operand is specified or if *input_file* is '-'.
 112712 See the INPUT FILES section.

112713 **INPUT FILES**

112714 The input file shall be a text file.

112715 **ENVIRONMENT VARIABLES**

112716 The following environment variables shall affect the execution of *uniq*:

112717 *LANG* Provide a default value for the internationalization variables that are unset or null.
 112718 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 112719 variables used to determine the values of locale categories.)

112720 *LC_ALL* If set to a non-empty string value, override the values of all the other
 112721 internationalization variables.

112722 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 112723 characters (for example, single-byte as opposed to multi-byte characters in
 112724 arguments and input files) and which characters constitute a <blank> in the
 112725 current locale.

112726 *LC_MESSAGES*

112727 Determine the locale that should be used to affect the format and contents of
 112728 diagnostic messages written to standard error.

112729 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

112730 **ASYNCHRONOUS EVENTS**

112731 Default.

112732 **STDOUT**

112733 The standard output shall be used if no *output_file* operand is specified, and shall be used if the
 112734 *output_file* operand is '-' and the implementation treats the '-' as meaning standard output.
 112735 Otherwise, the standard output shall not be used. See the OUTPUT FILES section.

112736 **STDERR**

112737 The standard error shall be used only for diagnostic messages.

112738 **OUTPUT FILES**

112739 If the *-c* option is specified, the output file shall be empty or each line shall be of the form:

112740 "%d %s", <number of duplicates>, <line>

112741 otherwise, the output file shall be empty or each line shall be of the form:

112742 "%s", <line>

112743 **EXTENDED DESCRIPTION**

112744 None.

112745 **EXIT STATUS**

112746 The following exit values shall be returned:

112747 0 The utility executed successfully.

112748 >0 An error occurred.

112749 **CONSEQUENCES OF ERRORS**

112750 Default.

112751 **APPLICATION USAGE**

112752 If the collating sequence of the current locale has a total ordering of all characters, the *sort* utility
 112753 can be used to cause repeated lines to be adjacent in the input file. If the collating sequence does
 112754 not have a total ordering of all characters, the *sort* utility should still do this but it might not. To
 112755 ensure that all duplicate lines are eliminated, and have the output sorted according the collating
 112756 sequence of the current locale, applications should use:

```
112757 LC_ALL=C sort -u | sort
```

112758 instead of:

```
112759 sort | uniq
```

112760 To remove duplicate lines based on whether they collate equally instead of whether they are
 112761 identical, applications should use:

```
112762 sort -u
```

112763 instead of:

```
112764 sort | uniq
```

112765 When using *uniq* to process pathnames, it is recommended that *LC_ALL*, or at least *LC_CTYPE*
 112766 and *LC_COLLATE*, are set to *POSIX* or *C* in the environment, since pathnames can contain byte
 112767 sequences that do not form valid characters in some locales, in which case the utility's behavior
 112768 would be undefined. In the *POSIX* locale each byte is a valid single-byte character, and therefore
 112769 this problem is avoided.

112770 **EXAMPLES**

112771 The following input file data (but flushed left) was used for a test series on *uniq*:

```
112772 #01 foo0 bar0 fool bar1
112773 #02 bar0 fool bar1 fool
112774 #03 foo0 bar0 fool bar1
112775 #04
112776 #05 foo0 bar0 fool bar1
112777 #06 foo0 bar0 fool bar1
112778 #07 bar0 fool bar1 foo0
```

112779 What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options
 112780 against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the
 112781 input data as a sequence of strings delimited by '\n'. Accordingly, for the *fields*th member of
 112782 the sequence, *uniq* interprets unique or repeated adjacent lines strictly relative to the *fields*+1th
 112783 member.

112784 1. This first example tests the line counting option, comparing each line of the input file data
 112785 starting from the second field:

```
112786 uniq -c -f 1 uniq_0I.t
112787     1 #01 foo0 bar0 fool bar1
112788     1 #02 bar0 fool bar1 fool
112789     1 #03 foo0 bar0 fool bar1
112790     1 #04
112791     2 #05 foo0 bar0 fool bar1
112792     1 #07 bar0 fool bar1 foo0
```

112793 The number '2', prefixing the fifth line of output, signifies that the *uniq* utility detected a
 112794 pair of repeated lines. Given the input data, this can only be true when *uniq* is run using
 112795 the **-f 1** option (which shall cause *uniq* to ignore the first field on each input line).

112796 2. The second example tests the option to suppress unique lines, comparing each line of the
112797 input file data starting from the second field:

```
112798     uniq -d -f 1 uniq_0I.t
112799     #05 foo0 bar0 fool bar1
```

112800 3. This test suppresses repeated lines, comparing each line of the input file data starting
112801 from the second field:

```
112802     uniq -u -f 1 uniq_0I.t
112803     #01 foo0 bar0 fool bar1
112804     #02 bar0 fool bar1 fool
112805     #03 foo0 bar0 fool bar1
112806     #04
112807     #07 bar0 fool bar1 foo0
```

112808 4. This suppresses unique lines, comparing each line of the input file data starting from the
112809 third character:

```
112810     uniq -d -s 2 uniq_0I.t
```

112811 In the last example, the *uniq* utility found no input matching the above criteria.

112812 RATIONALE

112813 Some historical implementations have limited lines to be 1 080 bytes in length, which does not
112814 meet the implied {LINE_MAX} limit.

112815 Earlier versions of this standard allowed the *-number* and *+number* options. These options are no
112816 longer specified by POSIX.1-2017 but may be present in some implementations.

112817 FUTURE DIRECTIONS

112818 None.

112819 SEE ALSO

112820 *comm*, *sort*

112821 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

112822 CHANGE HISTORY

112823 First released in Issue 2.

112824 Issue 6

112825 The obsolescent SYNOPSIS and associated text are removed.

112826 The normative text is reworded to avoid use of the term “must” for application requirements.

112827 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/40 is applied, adding *LC_COLLATE* to the
112828 ENVIRONMENT VARIABLES section, and changing “the application shall ensure that” in the
112829 OUTPUT FILES section.

112830 Issue 7

112831 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized
112832 as an option delimiter in the OPTIONS section.

112833 Austin Group Interpretation 1003.1-2001 #092 is applied.

112834 Austin Group Interpretation 1003.1-2001 #133 is applied, clarifying the behavior of the trailing
112835 <newline>.

112836 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112837 SD5-XCU-ERN-141 is applied, updating the EXAMPLES section.

112838
112839

POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0199 [963] and XCU/TC2-2008/0200 [663] are applied.

112840 **NAME**112841 unlink ‡call the *unlink()* function112842 **SYNOPSIS**112843 XSI unlink *file*112844 **DESCRIPTION**112845 The *unlink* utility shall perform the function call:112846 unlink(*file*);112847 A user may need appropriate privileges to invoke the *unlink* utility.112848 **OPTIONS**

112849 None.

112850 **OPERANDS**

112851 The following operands shall be supported:

112852 *file* The pathname of an existing file.112853 **STDIN**

112854 Not used.

112855 **INPUT FILES**

112856 Not used.

112857 **ENVIRONMENT VARIABLES**112858 The following environment variables shall affect the execution of *unlink*:112859 *LANG* Provide a default value for the internationalization variables that are unset or null.
112860 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
112861 variables used to determine the values of locale categories.)112862 *LC_ALL* If set to a non-empty string value, override the values of all the other
112863 internationalization variables.112864 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
112865 characters (for example, single-byte as opposed to multi-byte characters in
112866 arguments).112867 *LC_MESSAGES*112868 Determine the locale that should be used to affect the format and contents of
112869 diagnostic messages written to standard error.112870 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.112871 **ASYNCHRONOUS EVENTS**

112872 Default.

112873 **STDOUT**

112874 None.

112875 **STDERR**

112876 The standard error shall be used only for diagnostic messages.

112877 **OUTPUT FILES**

112878 None.

112879 **EXTENDED DESCRIPTION**

112880 None.

112881 **EXIT STATUS**

112882 The following exit values shall be returned:

112883 0 Successful completion.

112884 >0 An error occurred.

112885 **CONSEQUENCES OF ERRORS**

112886 Default.

112887 **APPLICATION USAGE**

112888 None.

112889 **EXAMPLES**

112890 None.

112891 **RATIONALE**

112892 None.

112893 **FUTURE DIRECTIONS**

112894 None.

112895 **SEE ALSO**112896 *link, rm*112897 XBD [Chapter 8](#) (on page 173)112898 XSH *unlink()*112899 **CHANGE HISTORY**

112900 First released in Issue 5.

112901 **NAME**

112902 uucp ‡system-to-system copy

112903 **SYNOPSIS**112904 UU `uucp [-cCdfjmr] [-n user] source-file... destination-file`112905 **DESCRIPTION**

112906 The *uucp* utility shall copy files named by the *source-file* argument to the *destination-file* argument.
 112907 The files named can be on local or remote systems.

112908 The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For
 112909 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and
 112910 filenames need not be portable to non-internationalized systems, and so on. Under these
 112911 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991
 112912 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used,
 112913 and that only characters defined in the portable filename character set be used for naming files.
 112914 The protocol for transfer of files is unspecified by POSIX.1-2017.

112915 Typical implementations of this utility require a communications line configured to use XBD
 112916 [Chapter 11](#) (on page 199), but other communications means may be used. On systems where
 112917 there are no available communications means (either temporarily or permanently), this utility
 112918 shall write an error message describing the problem and exit with a non-zero exit status.

112919 **OPTIONS**112920 The *uucp* utility shall conform to XBD [Section 12.2](#) (on page 216).

112921 The following options shall be supported:

- 112922 **-c** Do not copy local file to the spool directory for transfer to the remote machine
 112923 (default).
- 112924 **-C** Force the copy of local files to the spool directory for transfer.
- 112925 **-d** Make all necessary directories for the file copy (default).
- 112926 **-f** Do not make intermediate directories for the file copy.
- 112927 **-j** Write the job identification string to standard output. This job identification can be
 112928 used by *uustat* to obtain the status or terminate a job.
- 112929 **-m** Send mail to the requester when the copy is completed.
- 112930 **-n user** Notify *user* on the remote system that a file was sent.
- 112931 **-r** Do not start the file transfer; just queue the job.

112932 **OPERANDS**

112933 The following operands shall be supported:

112934 *destination-file, source-file*

112935 A pathname of a file to be copied to, or from, respectively. Either name can be a
 112936 pathname on the local machine, or can have the form:

112937 *system-name!pathname*

112938 where *system-name* is taken from a list of system names that *uucp* knows about.
 112939 The destination *system-name* can also be a list of names such as:

112940 *system-name!system-name!...!system-name!pathname*

112941 in which case, an attempt is made to send the file via the specified route to the

112942 destination. Care should be taken to ensure that intermediate nodes in the route
112943 are willing to forward information.

112944 The shell pattern matching notation characters '?', '*', and "[...]" appearing
112945 in *pathname* shall be expanded on the appropriate system.

112946 Pathnames can be one of:

- 112947 1. An absolute pathname.
- 112948 2. A pathname preceded by *~user* where *user* is a login name on the specified
112949 system and is replaced by that user's login directory. Note that if an invalid
112950 login is specified, the default is to the public directory (called *PUBDIR*; the
112951 actual location of *PUBDIR* is implementation-defined).
- 112952 3. A pathname preceded by *~/destination* where *destination* is appended to
112953 *PUBDIR*.

112954 **Note:** This destination is treated as a filename unless more than one file is being
112955 transferred by this request or the destination is already a directory. To
112956 ensure that it is a directory, follow the destination with a '/'. For
112957 example, *~/dan/* as the destination makes the directory **PUBDIR/dan** if it
112958 does not exist and puts the requested files in that directory.

- 112959 4. Anything else shall be prefixed by the current directory.

112960 If the result is an erroneous pathname for the remote system, the copy shall fail. If
112961 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

112962 The read, write, and execute permissions given by *uucp* are implementation-
112963 defined.

112964 STDIN

112965 Not used.

112966 INPUT FILES

112967 The files to be copied are regular files.

112968 ENVIRONMENT VARIABLES

112969 The following environment variables shall affect the execution of *uucp*:

112970 *LANG* Provide a default value for the internationalization variables that are unset or null.
112971 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
112972 variables used to determine the values of locale categories.)

112973 *LC_ALL* If set to a non-empty string value, override the values of all the other
112974 internationalization variables.

112975 *LC_COLLATE*

112976 Determine the locale for the behavior of ranges, equivalence classes, and multi-
112977 character collating elements within bracketed filename patterns.

112978 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
112979 characters (for example, single-byte as opposed to multi-byte characters in
112980 arguments and input files) and the behavior of character classes within bracketed
112981 filename patterns (for example, "[[:lower:]]*").

112982 *LC_MESSAGES*

112983 Determine the locale that should be used to affect the format and contents of
112984 diagnostic messages written to standard error, and informative messages written
112985 to standard output.

- 112986 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 112987 **ASYNCHRONOUS EVENTS**
- 112988 Default.
- 112989 **STDOUT**
- 112990 Not used.
- 112991 **STDERR**
- 112992 The standard error shall be used only for diagnostic messages.
- 112993 **OUTPUT FILES**
- 112994 The output files (which may be on other systems) are copies of the input files.
- 112995 If *-m* is used, mail files are modified.
- 112996 **EXTENDED DESCRIPTION**
- 112997 None.
- 112998 **EXIT STATUS**
- 112999 The following exit values shall be returned:
- 113000 0 Successful completion.
- 113001 >0 An error occurred.
- 113002 **CONSEQUENCES OF ERRORS**
- 113003 Default.
- 113004 **APPLICATION USAGE**
- 113005 This utility is part of the UUCP Utilities option and need not be supported by all
113006 implementations.
- 113007 The domain of remotely accessible files can (and for obvious security reasons usually should) be
113008 severely restricted.
- 113009 Note that the *'!*' character in addresses has to be escaped when using *cs**h* as a command
113010 interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary,
113011 but may be used.
- 113012 As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate
113013 system. On an internationalized system, this is done under the control of local settings of
113014 *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename
113015 patterns, as collation and typing rules may vary from one system to another. Also be aware that
113016 certain types of expression (that is, equivalence classes, character classes, and collating symbols)
113017 need not be supported on non-internationalized systems.
- 113018 **EXAMPLES**
- 113019 None.
- 113020 **RATIONALE**
- 113021 None.
- 113022 **FUTURE DIRECTIONS**
- 113023 None.
- 113024 **SEE ALSO**
- 113025 *mailx*, *uuencode*, *uustat*, *uux*
- 113026 XBD [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 216)

113027 **CHANGE HISTORY**

113028 First released in Issue 2.

113029 **Issue 6**

113030 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

113031 The UN margin codes and associated shading are removed from the *-C*, *-f*, *-j*, *-n*, and *-r*
113032 options in response to The Open Group Base Resolution bwg2001-003.

113033 **Issue 7**

113034 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

113035 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

113036 **NAME**

113037 uudecode ‡'decode a binary file

113038 **SYNOPSIS**113039 uudecode [-o *outfile*] [*file*]113040 **DESCRIPTION**

113041 The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data
 113042 created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data
 113043 compatible with one of the formats specified in *uuencode*, and attempt to create or overwrite the
 113044 file described by the data (or overridden by the **-o** option). The pathname shall be contained in
 113045 the data or specified by the **-o** option. The file access permission bits and contents for the file to
 113046 be produced shall be contained in that data. The mode bits of the created file (other than
 113047 standard output) shall be set from the file access permission bits contained in the data; that is,
 113048 other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect
 113049 the file being produced. If either of the *op* characters '+' and '-' (see *chmod*) are specified in
 113050 symbolic mode, the initial mode on which those operations are based is unspecified.

113051 If the pathname of the file resolves to an existing file and the user does not have write
 113052 permission on that file, *uudecode* shall terminate with an error. If the pathname of the file resolves
 113053 to an existing file and the user has write permission on that file, the existing file shall be
 113054 overwritten and, if possible, the mode bits of the file (other than standard output) shall be set as
 113055 described above; if the mode bits cannot be set, *uudecode* shall not treat this as an error.

113056 If the input data was produced by *uuencode* on a system with a different number of bits per byte
 113057 than on the target system, the results of *uudecode* are unspecified.

113058 **OPTIONS**113059 The *uudecode* utility shall conform to XBD [Section 12.2](#) (on page 216).

113060 The following option shall be supported by the implementation:

113061 **-o *outfile*** A pathname of a file that shall be used instead of any pathname contained in the
 113062 input data. Specifying an *outfile* option-argument of **/dev/stdout** shall indicate
 113063 standard output.

113064 **OPERANDS**

113065 The following operand shall be supported:

113066 *file* The pathname of a file containing the output of *uuencode*.113067 **STDIN**

113068 See the INPUT FILES section.

113069 **INPUT FILES**113070 The input files shall be files containing the output of *uuencode*.113071 **ENVIRONMENT VARIABLES**113072 The following environment variables shall affect the execution of *uudecode*:

113073 **LANG** Provide a default value for the internationalization variables that are unset or null.
 113074 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 113075 variables used to determine the values of locale categories.)

113076 **LC_ALL** If set to a non-empty string value, override the values of all the other
 113077 internationalization variables.

113078 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 113079 characters (for example, single-byte as opposed to multi-byte characters in
 113080 arguments and input files).

- 113081 *LC_MESSAGES*
- 113082 Determine the locale that should be used to affect the format and contents of
- 113083 diagnostic messages written to standard error.
- 113084 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 113085 **ASYNCHRONOUS EVENTS**
- 113086 Default.
- 113087 **STDOUT**
- 113088 If the file data header encoded by *uuencode* is `-` or `/dev/stdout`, or the `-o /dev/stdout` option
- 113089 overrides the file data, the standard output shall be in the same format as the file originally
- 113090 encoded by *uuencode*. Otherwise, the standard output shall not be used.
- 113091 **STDERR**
- 113092 The standard error shall be used only for diagnostic messages.
- 113093 **OUTPUT FILES**
- 113094 The output file shall be in the same format as the file originally encoded by *uuencode*.
- 113095 **EXTENDED DESCRIPTION**
- 113096 None.
- 113097 **EXIT STATUS**
- 113098 The following exit values shall be returned:
- 113099 0 Successful completion.
- 113100 >0 An error occurred.
- 113101 **CONSEQUENCES OF ERRORS**
- 113102 Default.
- 113103 **APPLICATION USAGE**
- 113104 The user who is invoking *uudecode* must have write permission on any file being created.
- 113105 The output of *uuencode* is essentially an encoded bit stream that is not cognizant of byte
- 113106 boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source,
- 113107 if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only
- 113108 data that is meaningful for such a transfer between architectures is generally character data.
- 113109 **EXAMPLES**
- 113110 None.
- 113111 **RATIONALE**
- 113112 Input files are not necessarily text files, as stated by an early proposal. Although the *uuencode*
- 113113 output is a text file, that output could have been wrapped within another file or mail message
- 113114 that is not a text file.
- 113115 The `-o` option is not historical practice, but was added at the request of WG15 so that the user
- 113116 could override the target pathname without having to edit the input data itself.
- 113117 In early drafts, the `[-o outfile]` option-argument allowed the use of `-` to mean standard output.
- 113118 The symbol `-` has only been used previously in POSIX.1-2017 as a standard input indicator. The
- 113119 standard developers did not wish to overload the meaning of `-` in this manner. The `/dev/stdout`
- 113120 concept exists on most modern systems. The `/dev/stdout` syntax does not refer to a new special
- 113121 file. It is just a magic cookie to specify standard output.

113122 **FUTURE DIRECTIONS**

113123 None.

113124 **SEE ALSO**113125 *chmod, umask, uuencode*113126 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)113127 **CHANGE HISTORY**

113128 First released in Issue 4.

113129 **Issue 6**

113130 This utility is marked as part of the User Portability Utilities option.

113131 The `-o outfile` option is added, as specified in the IEEE P1003.2b draft standard.

113132 The normative text is reworded to avoid use of the term “must” for application requirements.

113133 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/35 is applied, clarifying in the
113134 DESCRIPTION that the initial mode used if either of the *op* characters is '+' or '-' is
113135 unspecified.113136 **Issue 7**113137 The *uudecode* utility is moved from the User Portability Utilities option to the Base. User
113138 Portability Utilities is now an option for interactive utilities.

113139 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

113140 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0201 [635] is applied.

113141 **NAME**113142 uuencode *⚔* encode a binary file113143 **SYNOPSIS**113144 uuencode [-m] [*file*] *decode_pathname*113145 **DESCRIPTION**

113146 The *uuencode* utility shall write an encoded version of the named input file, or standard input if
 113147 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms
 113148 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal
 113149 or symbolic notation) of the input file and the *decode_pathname*, for re-creation of the file on
 113150 another system that conforms to this volume of POSIX.1-2017.

113151 **OPTIONS**113152 The *uuencode* utility shall conform to XBD [Section 12.2](#) (on page 216).

113153 The following option shall be supported by the implementation:

113154 **-m** Encode the output using the MIME Base64 algorithm described in STDOUT. If **-m**
 113155 is not specified, the historical algorithm described in STDOUT shall be used.

113156 **OPERANDS**

113157 The following operands shall be supported:

113158 *decode_pathname*

113159 The pathname of the file into which the *uudecode* utility shall place the decoded
 113160 file. Specifying a *decode_pathname* operand of **/dev/stdout** shall indicate that
 113161 *uudecode* is to use standard output. If there are characters in *decode_pathname* that
 113162 are not in the portable filename character set the results are unspecified.

113163 *file* A pathname of the file to be encoded.113164 **STDIN**

113165 See the INPUT FILES section.

113166 **INPUT FILES**

113167 Input files can be files of any type.

113168 **ENVIRONMENT VARIABLES**113169 The following environment variables shall affect the execution of *uuencode*:

113170 **LANG** Provide a default value for the internationalization variables that are unset or null.
 113171 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 113172 variables used to determine the values of locale categories.)

113173 **LC_ALL** If set to a non-empty string value, override the values of all the other
 113174 internationalization variables.

113175 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 113176 characters (for example, single-byte as opposed to multi-byte characters in
 113177 arguments and input files).

113178 **LC_MESSAGES**

113179 Determine the locale that should be used to affect the format and contents of
 113180 diagnostic messages written to standard error.

113181 **XSIX** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

113182 **ASYNCHRONOUS EVENTS**

113183 Default.

113184 **STDOUT**113185 **uuencode Base64 Algorithm**

113186 The standard output shall be a text file (encoded in the character set of the current locale) that
 113187 begins with the line:

113188 `"begin-base64Δ%sΔ\n", <mode>, <decode_pathname>`

113189 and ends with the line:

113190 `"====\n"`

113191 In both cases, the lines shall have no preceding or trailing <blank> characters.

113192 The encoding process represents 24-bit groups of input bits as output strings of four encoded
 113193 characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating
 113194 three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit
 113195 groups, each of which shall be translated into a single digit in the Base64 alphabet. When
 113196 encoding a bit stream via the Base64 encoding, the bit stream shall be presumed to be ordered
 113197 with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in
 113198 the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit
 113199 group is used as an index into an array of 64 printable characters, as shown in [Table 4-22](#).

113200 **Table 4-22** uuencode Base64 Values

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

113219 The character referenced by the index shall be placed in the output string.

113220 The output stream (encoded bytes) shall be represented in lines of no more than 76 characters
 113221 each. All line breaks or other characters not found in the table shall be ignored by decoding
 113222 software (see [uudecode](#)).

113223 Special processing shall be performed if fewer than 24 bits are available at the end of a message
 113224 or encapsulated part of a message. A full encoding quantum shall always be completed at the

113225 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall
 113226 be added (on the right) to form an integral number of 6-bit groups. Output character positions
 113227 that are not required to represent actual input data shall be set to the character '='. Since all
 113228 Base64 input is an integral number of octets, only the following cases can arise:

- 113229 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit
 113230 of encoded output shall be an integral multiple of 4 characters with no '=' padding.
- 113231 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded
 113232 output shall be three characters followed by one '=' padding character.
- 113233 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded
 113234 output shall be two characters followed by two '=' padding characters.

113235 A terminating "====" evaluates to nothing and denotes the end of the encoded data.

113236 uuencode Historical Algorithm

113237 The standard output shall be a text file (encoded in the character set of the current locale) that
 113238 begins with the line:

```
113239 "beginΔ%sΔ%s\n" <mode>, <decode_pathname>
```

113240 and ends with the line:

```
113241 "end\n"
```

113242 In both cases, the lines shall have no preceding or trailing <blank> characters.

113243 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input
 113244 and writes four characters of output by splitting the input at six-bit intervals into four octets,
 113245 containing data in the lower six bits only. These octets shall be converted to characters by adding
 113246 a value of 0x20 to each octet, so that each octet is in the range [0x20,0x5f], and then it shall be
 113247 assumed to represent a printable character in the ISO/IEC 646:1991 standard encoded character
 113248 set. It then shall be translated into the corresponding character codes for the codeset in use in the
 113249 current locale. (For example, the octet 0x41, representing 'A', would be translated to 'A' in the
 113250 current codeset, such as 0xc1 if it were EBCDIC.)

113251 Where the bits of two octets are combined, the least significant bits of the first octet shall be
 113252 shifted left and combined with the most significant bits of the second octet shifted right. Thus
 113253 the three octets *A*, *B*, *C* shall be converted into the four octets:

```
113254 0x20 + (( A >> 2 ) & 0x3F)
```

```
113255 0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
```

```
113256 0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
```

```
113257 0x20 + (( C ) & 0x3F)
```

113258 These octets then shall be translated into the local character set.

113259 Each encoded line contains a length character, equal to the number of characters to be decoded
 113260 plus 0x20 translated to the local character set as described above, followed by the encoded
 113261 characters. The maximum number of octets to be encoded on each line shall be 45.

113262 STDERR

113263 The standard error shall be used only for diagnostic messages.

113264 **OUTPUT FILES**

113265 None.

113266 **EXTENDED DESCRIPTION**

113267 None.

113268 **EXIT STATUS**

113269 The following exit values shall be returned:

113270 0 Successful completion.

113271 >0 An error occurred.

113272 **CONSEQUENCES OF ERRORS**

113273 Default.

113274 **APPLICATION USAGE**113275 The file is expanded by 35 percent (each three octets become four, plus control information)
113276 causing it to take longer to transmit.

113277 Since this utility is intended to create files to be used for data interchange between systems with
113278 possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991
113279 standard was chosen for a midpoint in the algorithm as a known reference point. The output
113280 from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991
113281 standard codeset, it might not be a text file (at least because the <newline> characters might not
113282 match), and the goal of creating a text file would be defeated. If this text file was then carried to
113283 another machine with the same codeset, it would be perfectly compatible with that system's
113284 *uudecode*. If it was transmitted over a mail system or sent to a machine with a different codeset,
113285 it is assumed that, as for every other text file, some translation mechanism would convert it (by
113286 the time it reached a user on the other system) into an appropriate codeset. This translation only
113287 makes sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard
113288 representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives,
113289 intermixed with other text files in the same codeset.

113290 **EXAMPLES**

113291 None.

113292 **RATIONALE**

113293 A new algorithm was added at the request of the international community to parallel work in
113294 RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding
113295 is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A
113296 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented
113297 per printable character. (The extra 65th character, '=', is used to signify a special processing
113298 function.)

113299 This subset has the important property that it is represented identically in all versions of the
113300 ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also
113301 represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not
113302 share this property, which is the reason that a second algorithm was added to the ISO POSIX-2
113303 standard.

113304 The string "====" was used for the termination instead of the end used in the original format
113305 because the latter is a string that could be valid encoded input.

113306 In an early draft, the *-m* option was named *-b* (for Base64), but it was renamed to reflect its
113307 relationship to the RFC 2045. A *-u* was also present to invoke the default algorithm, but since
113308 this was not historical practice, it was omitted as being unnecessary.

- 113309 See the RATIONALE section in *uuencode* for the derivation of the `/dev/stdout` symbol.
- 113310 **FUTURE DIRECTIONS**
- 113311 None.
- 113312 **SEE ALSO**
- 113313 *chmod, mailx, uuencode*
- 113314 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 113315 **CHANGE HISTORY**
- 113316 First released in Issue 4.
- 113317 **Issue 6**
- 113318 This utility is marked as part of the User Portability Utilities option.
- 113319 The Base64 algorithm and the ability to output to `/dev/stdout` are added as specified in the
- 113320 IEEE P1003.2b draft standard.
- 113321 **Issue 7**
- 113322 The *uuencode* utility is moved from the User Portability Utilities option to the Base. User
- 113323 Portability Utilities is now an option for interactive utilities.
- 113324 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

113325 **NAME**

113326 uustat — uucp status enquiry and job control

113327 **SYNOPSIS**113328 UU uustat [-q|-k *jobid*|-r *jobid*]113329 uustat [-s *system*] [-u *user*]113330 **DESCRIPTION**113331 The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or
113332 provide general status on *uucp* connections to other systems.113333 When no options are given, *uustat* shall write to standard output the status of all *uucp* requests
113334 issued by the current user.113335 Typical implementations of this utility require a communications line configured to use XBD
113336 [Chapter 11](#) (on page 199), but other communications means may be used. On systems where
113337 there are no available communications means (either temporarily or permanently), this utility
113338 shall write an error message describing the problem and exit with a non-zero exit status.113339 **OPTIONS**113340 The *uustat* utility shall conform to XBD [Section 12.2](#) (on page 216).

113341 The following options shall be supported:

113342 **-q** Write the jobs queued for each machine.113343 **-k *jobid*** Kill the *uucp* request whose job identification is *jobid*. The application shall ensure
113344 that the killed *uucp* request belongs to the person invoking *uustat* unless that user
113345 has appropriate privileges.113346 **-r *jobid*** Rejuvenate *jobid*. The files associated with *jobid* are touched so that their
113347 modification time is set to the current time. This prevents the cleanup program
113348 from deleting the job until the jobs modification time reaches the limit imposed by
113349 the program.113350 **-s *system*** Write the status of all *uucp* requests for remote system *system*.113351 **-u *user*** Write the status of all *uucp* requests issued by *user*.113352 **OPERANDS**

113353 None.

113354 **STDIN**

113355 Not used.

113356 **INPUT FILES**

113357 None.

113358 **ENVIRONMENT VARIABLES**113359 The following environment variables shall affect the execution of *uustat*:113360 **LANG** Provide a default value for the internationalization variables that are unset or null.
113361 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
113362 variables used to determine the values of locale categories.)113363 **LC_ALL** If set to a non-empty string value, override the values of all the other
113364 internationalization variables.

- 113365 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
113366 characters (for example, single-byte as opposed to multi-byte characters in
113367 arguments).
- 113368 *LC_MESSAGES*
113369 Determine the locale that should be used to affect the format and contents of
113370 diagnostic messages written to standard error, and informative messages written
113371 to standard output.
- 113372 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 113373 **ASYNCHRONOUS EVENTS**
113374 Default.
- 113375 **STDOUT**
113376 The standard output shall consist of information about each job selected, in an unspecified
113377 format. The information shall include at least the job ID, the user ID or name, and the remote
113378 system name.
- 113379 **STDERR**
113380 The standard error shall be used only for diagnostic messages.
- 113381 **OUTPUT FILES**
113382 None.
- 113383 **EXTENDED DESCRIPTION**
113384 None.
- 113385 **EXIT STATUS**
113386 The following exit values shall be returned:
113387 0 Successful completion.
113388 >0 An error occurred.
- 113389 **CONSEQUENCES OF ERRORS**
113390 Default.
- 113391 **APPLICATION USAGE**
113392 This utility is part of the UUCP Utilities option and need not be supported by all
113393 implementations.
- 113394 **EXAMPLES**
113395 None.
- 113396 **RATIONALE**
113397 None.
- 113398 **FUTURE DIRECTIONS**
113399 None.
- 113400 **SEE ALSO**
113401 *uucp*
113402 XBD [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 216)
- 113403 **CHANGE HISTORY**
113404 First released in Issue 2.

113405 **Issue 6**

113406 The normative text is reworded to avoid use of the term “must” for application requirements.

113407 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

113408 The UN margin code and associated shading are removed from the *-q* option in response to The
113409 Open Group Base Resolution bwg2001-003.

113410 **Issue 7**

113411 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

113412 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

113413 **NAME**

113414 uux — remote command execution

113415 **SYNOPSIS**113416 UU `uux [-jnp] command-string`113417 **DESCRIPTION**

113418 The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see
 113419 [Section 2.9](#), on page 2365) on a specified system, and then send the standard output of the
 113420 command to a file on a specified system. Only the first command of a pipeline can have a *system-*
 113421 *name!* prefix. All other commands in the pipeline shall be executed on the system of the first
 113422 command.

113423 The following restrictions are applicable to the shell pipeline processed by *uux*:

113424 In gathering files from different systems, pathname expansion shall not be performed by
 113425 *uux*. Thus, a request such as:

113426 `uux "c99 remsys!~/*.c"`

113427 would attempt to copy the file named literally `*.c` to the local system.

113428 The redirection operators "`>>`", "`<<`", "`>|`", and "`>&`" shall not be accepted. Any use of
 113429 these redirection operators shall cause this utility to write an error message describing the
 113430 problem and exit with a non-zero exit status.

113431 The reserved word `!` cannot be used at the head of the pipeline to modify the exit status.
 113432 (See the *command-string* operand description below.)

113433 Alias substitution shall not be performed.

113434 A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by
 113435 *~name* (which is replaced by the corresponding login directory), a pathname specified as *~/dest*
 113436 (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is
 113437 implementation-defined), or a simple filename (which is prefixed by *uux* with the current
 113438 directory). See *uucp* for the details.

113439 The execution of commands on remote systems shall take place in an execution directory known
 113440 to the *uucp* system. All files required for the execution shall be put into this directory unless they
 113441 already reside on that machine. Therefore, the application shall ensure that non-local filenames
 113442 (without path or machine reference) are unique within the *uux* request.

113443 The *uux* utility shall attempt to get all files to the execution system. For files that are output files,
 113444 the application shall ensure that the filename is escaped using parentheses.

113445 The remote system shall notify the user by mail if the requested command on the remote system
 113446 was disallowed or the files were not accessible. This notification can be turned off by the `-n`
 113447 option.

113448 Typical implementations of this utility require a communications line configured to use XBD
 113449 [Chapter 11](#) (on page 199), but other communications means may be used. On systems where
 113450 there are no available communications means (either temporarily or permanently), this utility
 113451 shall write an error message describing the problem and exit with a non-zero exit status.

113452 The *uux* utility cannot guarantee support for all character encodings in all circumstances. For
 113453 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and
 113454 filenames need not be portable to non-internationalized systems, and so on. Under these
 113455 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991
 113456 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used

113457 and that only characters defined in the portable filename character set be used for naming files.

113458 **OPTIONS**

113459 The *uux* utility shall conform to XBD [Section 12.2](#) (on page 216).

113460 The following options shall be supported:

113461 **-j** Write the job identification string to standard output. This job identification can be
113462 used by *uustat* to obtain the status or terminate a job.

113463 **-n** Do not notify the user if the command fails.

113464 **-p** Make the standard input to *uux* the standard input to the *command-string*.

113465 **OPERANDS**

113466 The following operand shall be supported:

113467 *command-string*

113468 A string made up of one or more arguments that are similar to normal command
113469 arguments, except that the command and any filenames can be prefixed by *system-*
113470 *name!*. A null *system-name* shall be interpreted as the local system.

113471 **STDIN**

113472 The standard input shall not be used unless the **-** or **-p** option is specified; in those cases, the
113473 standard input shall be made the standard input of the *command-string*.

113474 **INPUT FILES**

113475 Input files shall be selected according to the contents of *command-string*.

113476 **ENVIRONMENT VARIABLES**

113477 The following environment variables shall affect the execution of *uux*:

113478 **LANG** Provide a default value for the internationalization variables that are unset or null.
113479 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
113480 variables used to determine the values of locale categories.)

113481 **LC_ALL** If set to a non-empty string value, override the values of all the other
113482 internationalization variables.

113483 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
113484 characters (for example, single-byte as opposed to multi-byte characters in
113485 arguments).

113486 **LC_MESSAGES**

113487 Determine the locale that should be used to affect the format and contents of
113488 diagnostic messages written to standard error.

113489 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

113490 **ASYNCHRONOUS EVENTS**

113491 Default.

113492 **STDOUT**

113493 The standard output shall not be used unless the **-j** option is specified; in that case, the job
113494 identification string shall be written to standard output in the following format:

113495 "%s\n", <*jobid*>

113496 **STDERR**

113497 The standard error shall be used only for diagnostic messages.

113498 **OUTPUT FILES**

113499 Output files shall be created or written, or both, according to the contents of *command-string*.

113500 If **-n** is not used, mail files shall be modified following any command or file-access failures on
113501 the remote system.

113502 **EXTENDED DESCRIPTION**

113503 None.

113504 **EXIT STATUS**

113505 The following exit values shall be returned:

113506 0 Successful completion.

113507 >0 An error occurred.

113508 **CONSEQUENCES OF ERRORS**

113509 Default.

113510 **APPLICATION USAGE**

113511 This utility is part of the UUCP Utilities option and need not be supported by all
113512 implementations.

113513 Note that, for security reasons, many installations limit the list of commands executable on
113514 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail
113515 via *uux*.

113516 Any characters special to the command interpreter should be quoted either by quoting the entire
113517 *command-string* or quoting the special characters as individual arguments.

113518 As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are
113519 expanded on the appropriate local system. This is done under the control of local settings of
113520 *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename
113521 patterns, as collation and typing rules may vary from one system to another. Also be aware that
113522 certain types of expression (that is, equivalence classes, character classes, and collating symbols)
113523 need not be supported on non-internationalized systems.

113524 **EXAMPLES**

113525 1. The following command gets **file1** from system **a** and **file2** from system **b**, executes *diff* on
113526 the local system, and puts the results in **file.diff** in the local *PUBDIR* directory. (*PUBDIR*
113527 is the *uucp* public directory on the local system.)

```
113528 uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"
```

113529 2. The following command fails because *uux* places all files copied to a system in the same
113530 working directory. Although the files **xyz** are from two different systems, their filenames
113531 are the same and conflict.

```
113532 uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"
```

113533 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the
113534 file local to system **a** is not copied to the working directory, and hence does not conflict
113535 with the file from system **c**.

```
113536 uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"
```

113537 **RATIONALE**

113538 None.

113539 **FUTURE DIRECTIONS**

113540 None.

113541 **SEE ALSO**113542 [Chapter 2](#) (on page 2345), [uucp](#), [uuencode](#), [uustat](#)113543 [XBD Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199), [Section 12.2](#) (on page 216)113544 **CHANGE HISTORY**

113545 First released in Issue 2.

113546 **Issue 6**

113547 The obsolescent SYNOPSIS is removed.

113548 The normative text is reworded to avoid use of the term “must” for application requirements.

113549 The UN margin code and associated shading are removed from the `-j` option in response to The
113550 Open Group Base Resolution bwg2001-003.113551 **Issue 7**

113552 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

113553 **NAME**

113554 val ‡validate SCCS files(DEVELOPMENT)

113555 **SYNOPSIS**

```
113556 XSI val -
113557 val [-s] [-m name] [-r SID] [-y type] file...
```

113558 **DESCRIPTION**

113559 The *val* utility shall determine whether the specified *file* is an SCCS file meeting the
 113560 characteristics specified by the options.

113561 **OPTIONS**

113562 The *val* utility shall conform to XBD [Section 12.2](#) (on page 216), except that the usage of the '-'
 113563 operand is not strictly as intended by the guidelines (that is, reading options and operands from
 113564 standard input).

113565 The following options shall be supported:

- 113566 **-m name** Specify a *name*, which is compared with the SCCS %M% keyword in *file*; see *get*.
- 113567 **-r SID** Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be
 113568 made to determine whether the *SID* is ambiguous (for example, **-r 1** is ambiguous
 113569 because it physically does not exist but implies 1.1, 1.2, and so on, which may exist)
 113570 or invalid (for example, **-r 1.0** or **-r 1.1.0** are invalid because neither case can exist
 113571 as a valid delta number). If the *SID* is valid and not ambiguous, a check shall be
 113572 made to determine whether it actually exists.
- 113573 **-s** Silence the diagnostic message normally written to standard output for any error
 113574 that is detected while processing each named file on a given command line.
- 113575 **-y type** Specify a *type*, which shall be compared with the SCCS %Y% keyword in *file*; see
 113576 *get*.

113577 **OPERANDS**

113578 The following operands shall be supported:

- 113579 *file* A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is
 113580 '-', the standard input shall be read: each line shall be independently processed
 113581 as if it were a command line argument list. (However, the line is not subjected to
 113582 any of the shell word expansions, such as parameter expansion or quote removal.)

113583 **STDIN**

113584 The standard input shall be a text file used only when the *file* operand is specified as '-'.

113585 **INPUT FILES**

113586 Any SCCS files processed shall be files of an unspecified format.

113587 **ENVIRONMENT VARIABLES**

113588 The following environment variables shall affect the execution of *val*:

- 113589 **LANG** Provide a default value for the internationalization variables that are unset or null.
 113590 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 113591 variables used to determine the values of locale categories.)
- 113592 **LC_ALL** If set to a non-empty string value, override the values of all the other
 113593 internationalization variables.

113594 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 113595 characters (for example, single-byte as opposed to multi-byte characters in
 113596 arguments and input files).

113597 *LC_MESSAGES*
 113598 Determine the locale that should be used to affect the format and contents of
 113599 diagnostic messages written to standard error, and informative messages written
 113600 to standard output.

113601 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

113602 **ASYNCHRONOUS EVENTS**

113603 Default.

113604 **STDOUT**

113605 The standard output shall consist of informative messages about either:

113606 1. Each file processed

113607 2. Each command line read from standard input

113608 If the standard input is not used, for each *file* operand yielding a discrepancy, the output line
 113609 shall have the following format:

113610 "%s: %s\n", <pathname>, <unspecified string>

113611 If the standard input is used, for each input line yielding a discrepancy, the output shall have the
 113612 following format:

113613 "%s\n\n %s: %s\n", <input>, <pathname>, <unspecified string>

113614 where <input> is the input line minus its terminating <newline>.

113615 **STDERR**

113616 Not used.

113617 **OUTPUT FILES**

113618 None.

113619 **EXTENDED DESCRIPTION**

113620 None.

113621 **EXIT STATUS**

113622 The 8-bit code returned by *val* shall be a disjunction of the possible errors; that is, it can be
 113623 interpreted as a bit string where set bits are interpreted as follows:

113624 0x80 = Missing file argument.

113625 0x40 = Unknown or duplicate option.

113626 0x20 = Corrupted SCCS file.

113627 0x10 = Cannot open file or file not SCCS.

113628 0x08 = *SID* is invalid or ambiguous.

113629 0x04 = *SID* does not exist.

113630 0x02 = %Y%, -y mismatch.

113631 0x01 = %M%, -m mismatch.

113632 Note that *val* can process two or more files on a given command line and can process multiple
 113633 command lines (when reading the standard input). In these cases an aggregate code shall be
 113634 returned: a logical OR of the codes generated for each command line and file processed.

113635 CONSEQUENCES OF ERRORS

113636 Default.

113637 APPLICATION USAGE

113638 Since the *val* exit status sets the 0x80 bit, shell applications checking "\$?" cannot tell if it
113639 terminated due to a missing file argument or receipt of a signal.

113640 EXAMPLES

113641 In a directory with three SCCS files—*s.x* (of *t* type `text`), *s.y*, and *s.z* (a corrupted file)—the
113642 following command could produce the output shown:

```
113643 val - <<EOF
113644 -y source s.x
113645 -m y s.y
113646 s.z
113647 EOF
113648 -y source s.x

113649 s.x: %Y%, -y mismatch
113650 s.z
113651 s.z: corrupted SCCS file
```

113652 RATIONALE

113653 None.

113654 FUTURE DIRECTIONS

113655 None.

113656 SEE ALSO

113657 *admin, delta, get, prs*

113658 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

113659 CHANGE HISTORY

113660 First released in Issue 2.

113661 Issue 6

113662 The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT
113663 STATUS.

113664 Issue 7

113665 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

113666 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0147 [416] and XCU/TC1-2008/0148
113667 [416] are applied.

113668 **NAME**

113669 vi — screen-oriented (visual) display editor

113670 **SYNOPSIS**113671 UP `vi [-rR] [-c command] [-t tagstring] [-w size] [file...]`113672 **DESCRIPTION**113673 This utility shall be provided on systems that both support the User Portability Utilities option
113674 and define the POSIX2_CHAR_TERM symbol. On other systems it is optional.113675 The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the
113676 editor are described in POSIX.1-2017; see the line editor *ex* for additional editing capabilities
113677 used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex* commands from
113678 within *vi*.113679 This reference page uses the term *edit buffer* to describe the current working text. No specific
113680 implementation is implied by this term. All editing changes are performed on the edit buffer,
113681 and no changes to it shall affect any file until an editor command writes the file.113682 When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to
113683 the editing buffer shall be reflected in the screen display; the position of the cursor on the screen
113684 shall indicate the position within the editing buffer.113685 Certain terminals do not have all the capabilities necessary to support the complete *vi* definition.
113686 When these commands cannot be supported on such terminals, this condition shall not produce
113687 an error message such as “not an editor command” or report a syntax error. The implementation
113688 may either accept the commands and produce results on the screen that are the result of an
113689 unsuccessful attempt to meet the requirements of this volume of POSIX.1-2017 or report an error
113690 describing the terminal-related deficiency.113691 **OPTIONS**113692 The *vi* utility shall conform to XBD [Section 12.2](#) (on page 216), except that '+' may be
113693 recognized as an option delimiter as well as '-'.
113694 The following options shall be supported:

113694 The following options shall be supported:

113695 `-c command` See the *ex* command description of the `-c` option.113696 `-r` See the *ex* command description of the `-r` option.113697 `-R` See the *ex* command description of the `-R` option.113698 `-t tagstring` See the *ex* command description of the `-t` option.113699 `-w size` See the *ex* command description of the `-w` option.113700 **OPERANDS**113701 See the OPERANDS section of the *ex* command for a description of the operands supported by
113702 the *vi* command.113703 **STDIN**113704 If standard input is not a terminal device, the results are undefined. The standard input consists
113705 of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.113706 If a read from the standard input returns an error, or if the editor detects an end-of-file condition
113707 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

113708 **INPUT FILES**

113709 See the INPUT FILES section of the *ex* command for a description of the input files supported by
113710 the *vi* command.

113711 **ENVIRONMENT VARIABLES**

113712 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables
113713 that affect the execution of the *vi* command.

113714 **ASYNCHRONOUS EVENTS**

113715 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the
113716 execution of the *vi* command.

113717 **STDOUT**

113718 If standard output is not a terminal device, undefined results occur.

113719 Standard output may be used for writing prompts to the user, for informational messages, and
113720 for writing lines from the file.

113721 **STDERR**

113722 If standard output is not a terminal device, undefined results occur.

113723 The standard error shall be used only for diagnostic messages.

113724 **OUTPUT FILES**

113725 See the OUTPUT FILES section of the *ex* command for a description of the output files
113726 supported by the *vi* command.

113727 **EXTENDED DESCRIPTION**

113728 If the terminal does not have the capabilities necessary to support an unspecified portion of the
113729 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after
113730 initialization, *vi* shall be in command mode; text input mode can be entered by one of several
113731 commands used to insert or change text. In text input mode, <ESC> can be used to return to
113732 command mode; other uses of <ESC> are described later in this section; see [Terminate
113733 Command or Input Mode](#) (on page 3385).

113734 **Initialization in *ex* and *vi***

113735 See [Initialization in *ex* and *vi*](#) (on page 2701) for a description of *ex* and *vi* initialization for the *vi*
113736 utility.

113737 **Command Descriptions in *vi***

113738 The following symbols are used in this reference page to represent arguments to commands.

113739 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;
113740 see [Command Descriptions in *ex*](#) (on page 2710).

113741 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]
113742 preceding the command name, they can be specified in either order.

113743 *count* A positive integer used as an optional argument to most commands, either to give a
113744 repeat count or as a size. This argument is optional and shall default to 1 unless
113745 otherwise specified.

113746 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,
113747 <control>-], %, &, ^, D, m, M, Q, u, U, and ZZ do not have *count* as an optional
113748 argument. Regardless, it shall not be an error to specify a *count* to these commands, and
113749 any specified *count* shall be ignored.

113750 *motion* An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used
 113751 to indicate the region of text that shall be affected by the command. The motion can be
 113752 either one of the command characters repeated or one of several other *vi* commands
 113753 (listed in the following table). Each of the applicable commands specifies the region of
 113754 text matched by repeating the command; each command that can be used as a motion
 113755 command specifies the region of text it affects.

113756 Commands that take *motion* arguments operate on either lines or characters, depending
 113757 on the circumstances. When operating on lines, all lines that fall partially or wholly
 113758 within the text region specified for the command shall be affected. When operating on
 113759 characters, only the exact characters in the specified text region shall be affected. Each
 113760 motion command specifies this individually.

113761 When commands that may be motion commands are not used as motion commands,
 113762 they shall set the current position to the current line and column as specified.

113763 The following commands shall be valid cursor motion commands:

113764	<apostrophe>	(-	j	H	
113765	<carriage-return>)	\$	k	L	
113766	<comma>	[[%	l	M
113767	<control>-H]]	_	n	N
113768	<control>-N	{	;	t	T	
113769	<control>-P	}	?	w	W	
113770	<grave-accent>	^	b	B		
113771	<newline>	+	e	E		
113772	<space>		f	F		
113773	<zero>	/	h	G		

113774 Any *count* that is specified to a command that has an associated motion command shall
 113775 be applied to the motion command. If a *count* is applied to both the command and its
 113776 associated motion command, the effect shall be multiplicative.

113777 The following symbols are used in this section to specify locations in the edit buffer:

113778 *current character*

113779 The character that is currently indicated by the cursor.

113780 *end of a line*

113781 The point located between the last non-<newline> (if any) and the terminating
 113782 <newline> of a line. For an empty line, this location coincides with the beginning of the
 113783 line.

113784 *end of the edit buffer*

113785 The location corresponding to the end of the last line in the edit buffer.

113786 The following symbols are used in this section to specify command actions:

113787 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

- 113788 1. A maximal sequence of non-<blank> characters preceded and followed by
 113789 <blank> characters or the beginning or end of a line or the edit buffer
- 113790 2. One or more sequential blank lines
- 113791 3. The first character in the edit buffer

- 113792 4. The last non-<newline> in the edit buffer
- 113793 *word* In the POSIX locale, *vi* shall recognize five kinds of words:
- 113794 1. A maximal sequence of letters, digits, and underscores, delimited at both ends
- 113795 by:
- 113796 ‡ ~~l~~ Characters other than letters, digits, or underscores
- 113797 ‡ ~~l~~ Beginning or end of a line
- 113798 ‡ ~~l~~ Beginning or end of the edit buffer
- 113799 2. A maximal sequence of characters other than letters, digits, underscores, or
- 113800 <blank> characters, delimited at both ends by:
- 113801 ‡ ~~l~~ Letter, digit, underscore
- 113802 ‡ ~~l~~ <blank> characters
- 113803 ‡ ~~l~~ Beginning or end of a line
- 113804 ‡ ~~l~~ Beginning or end of the edit buffer
- 113805 3. One or more sequential blank lines
- 113806 4. The first character in the edit buffer
- 113807 5. The last non-<newline> in the edit buffer
- 113808 *section boundary*
- 113809 A *section boundary* is one of the following:
- 113810 1. A line whose first character is a <form-feed>
- 113811 2. A line whose first character is an open curly brace (' { ')
- 113812 3. A line whose first character is a <period> and whose second and third characters
- 113813 match a two-character pair in the **sections** edit option (see *ex*)
- 113814 4. A line whose first character is a <period> and whose only other character
- 113815 matches the first character of a two-character pair in the **sections** edit option,
- 113816 where the second character of the two-character pair is a <space>
- 113817 5. The first line of the edit buffer
- 113818 6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a
- 113819]] or } command; otherwise, the last non-<newline> of the last line of the edit
- 113820 buffer
- 113821 *paragraph boundary*
- 113822 A *paragraph boundary* is one of the following:
- 113823 1. A section boundary
- 113824 2. A line whose first character is a <period> and whose second and third characters
- 113825 match a two-character pair in the **paragraphs** edit option (see *ex*)
- 113826 3. A line whose first character is a <period> and whose only other character
- 113827 matches the first character of a two-character pair in the *paragraphs* edit option,
- 113828 where the second character of the two-character pair is a <space>

- 113829 4. One or more sequential blank lines
- 113830 *remembered search direction*
- 113831 See the description of *remembered search direction* in *ex*.
- 113832 *sentence boundary*
- 113833 A *sentence boundary* is one of the following:
- 113834 1. A paragraph boundary
 - 113835 2. The first non-<blank> that occurs after a paragraph boundary
 - 113836 3. The first non-<blank> that occurs after a <period> (' . '), <exclamation-mark>
 - 113837 (' ! '), or <question-mark> (' ? '), followed by two <space> characters or the end
 - 113838 of a line; any number of closing parenthesis (') '), closing brackets ('] '),
 - 113839 double-quote (' " '), or single-quote (<apostrophe>) characters can appear
 - 113840 between the punctuation mark and the two <space> characters or end-of-line
- 113841 In the remainder of the description of the *vi* utility, the term “buffer line” refers to a line in the
- 113842 edit buffer and the term “display line” refers to the line or lines on the display screen used to
- 113843 display one buffer line. The term “current line” refers to a specific “buffer line”.
- 113844 If there are display lines on the screen for which there are no corresponding buffer lines because
- 113845 they correspond to lines that would be after the end of the file, they shall be displayed as a single
- 113846 <tilde> (' ~ ') character, plus the terminating <newline>.
- 113847 The last line of the screen shall be used to report errors or display informational messages. It
- 113848 shall also be used to display the input for “line-oriented commands” (/ , ? , : , and !). When a line-
- 113849 oriented command is executed, the editor shall enter text input mode on the last line on the
- 113850 screen, using the respective command characters as prompt characters. (In the case of the !
- 113851 command, the associated motion shall be entered by the user before the editor enters text input
- 113852 mode.) The line entered by the user shall be terminated by a <newline>, a
- 113853 non-<control>-V-escaped <carriage-return>, or unescaped <ESC>. It is unspecified if more
- 113854 characters than require a display width minus one column number of screen columns can be
- 113855 entered.
- 113856 If any command is executed that overwrites a portion of the screen other than the last line of the
- 113857 screen (for example, the *ex* **suspend** or ! commands), other than the *ex* **shell** command, the user
- 113858 shall be prompted for a character before the screen is refreshed and the edit session continued.
- 113859 <tab> characters shall take up the number of columns on the screen set by the **tabstop** edit
- 113860 option (see *ex*), unless there are less than that number of columns before the display margin that
- 113861 will cause the displayed line to be folded; in this case, they shall only take up the number of
- 113862 columns up to that boundary.
- 113863 The cursor shall be placed on the current line and relative to the current column as specified by
- 113864 each command described in the following sections.
- 113865 In open mode, if the current line is not already displayed, then it shall be displayed.
- 113866 In visual mode, if the current line is not displayed, then the lines that are displayed shall be
- 113867 expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be
- 113868 displayed. If the screen is redrawn, no more than the number of display lines specified by the
- 113869 value of the **window** edit option shall be displayed (unless the current line cannot be completely
- 113870 displayed in the number of display lines specified by the **window** edit option) and the current
- 113871 line shall be positioned as close to the center of the displayed lines as possible (within the
- 113872 constraints imposed by the distance of the line from the beginning or end of the edit buffer). If
- 113873 the current line is before the first line in the display and the screen is scrolled, an unspecified

113874 portion of the current line shall be placed on the first line of the display. If the current line is after
 113875 the last line in the display and the screen is scrolled, an unspecified portion of the current line
 113876 shall be placed on the last line of the display.

113877 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into
 113878 the lines at the bottom of the display that are available for its presentation, the editor may choose
 113879 not to display any portion of the line. The lines of the display that do not contain text from the
 113880 edit buffer for this reason shall each consist of a single '@' character.

113881 In visual mode, the editor may choose for unspecified reasons to not update lines in the display
 113882 to correspond to the underlying edit buffer text. The lines of the display that do not correctly
 113883 correspond to text from the edit buffer for this reason shall consist of a single '@' character (plus
 113884 the terminating <newline>), and the <control>-R command shall cause the editor to update the
 113885 screen to correctly represent the edit buffer.

113886 Open and visual mode commands that set the current column set it to a column position in the
 113887 display, and not a character position in the line. In this case, however, the column position in the
 113888 display shall be calculated for an infinite width display; for example, the column related to a
 113889 character that is part of a line that has been folded onto additional screen lines will be offset from
 113890 the display line column where the buffer line begins, not from the beginning of a particular
 113891 display line.

113892 The display cursor column in the display is based on the value of the current column, as follows,
 113893 with each rule applied in turn:

- 113894 1. If the current column is after the last display line column used by the displayed line, the
 113895 display cursor column shall be set to the last display line column occupied by the last
 113896 non-<newline> in the current line; otherwise, the display cursor column shall be set to the
 113897 current column.
- 113898 2. If the character of which some portion is displayed in the display line column specified by
 113899 the display cursor column requires more than a single display line column:
 - 113900 a. If in text input mode, the display cursor column shall be adjusted to the first
 113901 display line column in which any portion of that character is displayed.
 - 113902 b. Otherwise, the display cursor column shall be adjusted to the last display line
 113903 column in which any portion of that character is displayed.

113904 The current column shall not be changed by these adjustments to the display cursor column.

113905 If an error occurs during the parsing or execution of a *vi* command:

113906 The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for
 113907 example, the current line and column) shall not be further modified.

113908 Unless otherwise specified by the following command sections, it is unspecified whether
 113909 an informational message shall be displayed.

113910 Any partially entered *vi* command shall be discarded.

113911 If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion
 113912 shall be discarded, except as otherwise specified by the **map** command (see *ex*).

113913 If the *vi* command resulted from the execution of a buffer, no further commands caused by
 113914 the execution of the buffer shall be executed.

113915 **Page Backwards**113916 *Synopsis:* [count] <control>-B113917 If in open mode, the <control>-B command shall behave identically to the **z** command.
113918 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.113919 If the **window** edit option is less than 3, display a screen where the last line of the display shall
113920 be some portion of:113921 *(current first line) -1*

113922 otherwise, display a screen where the first line of the display shall be some portion of:

113923 *(current first line) - count x ((window edit option) -2)*113924 If this calculation would result in a line that is before the first line of the edit buffer, the first line
113925 of the display shall display some portion of the first line of the edit buffer.113926 *Current line:* If no lines from the previous display remain on the screen, set to the last line of the
113927 display; otherwise, set to *(line - the number of new lines displayed on this screen)*.113928 *Current column:* Set to non-<blank>.113929 **Scroll Forward**113930 *Synopsis:* [count] <control>-D

113931 If the current line is the last line of the edit buffer, it shall be an error.

113932 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D
113933 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*
113934 shall default to the value of the **scroll** edit option.113935 If in open mode, write lines starting with the line after the current line, until *count* lines or the
113936 last line of the file have been written.113937 *Current line:* If the current line + *count* is past the last line of the edit buffer, set to the last line of
113938 the edit buffer; otherwise, set to the current line + *count*.113939 *Current column:* Set to non-<blank>.113940 **Scroll Forward by Line**113941 *Synopsis:* [count] <control>-E

113942 Display the line count lines after the last line currently displayed.

113943 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines
113944 after the last line currently displayed, the last line of the display shall display some portion of
113945 the last line of the edit buffer.113946 *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first
113947 line displayed.113948 *Current column:* Unchanged.

113949 **Page Forward**

113950 *Synopsis:* [*count*] <control>-F

113951 If in open mode, the <control>-F command shall behave identically to the **z** command.
113952 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.

113953 If the **window** edit option is less than 3, display a screen where the first line of the display shall
113954 be some portion of:

113955 (*current last line*) +1

113956 otherwise, display a screen where the first line of the display shall be some portion of:

113957 (*current first line*) + *count* x ((*window edit option*) -2)

113958 If this calculation would result in a line that is after the last line of the edit buffer, the last line of
113959 the display shall display some portion of the last line of the edit buffer.

113960 *Current line:* If no lines from the previous display remain on the screen, set to the first line of the
113961 display; otherwise, set to (*line* + the number of new lines displayed on this screen).

113962 *Current column:* Set to non-<blank>.

113963 **Display Information**

113964 *Synopsis:* <control>-G

113965 This command shall be equivalent to the *ex file* command.

113966 **Move Cursor Backwards**

113967 *Synopsis:* [*count*] <control>-H

113968 [*count*] h

113969 the current *erase* character (see *stty*)

113970 If there are no characters before the current character on the current line, it shall be an error. If
113971 there are less than *count* previous characters on the current line, *count* shall be adjusted to the
113972 number of previous characters on the line.

113973 If used as a motion command:

113974 1. The text region shall be from the character before the starting cursor up to and including
113975 the *count*th character before the starting cursor.

113976 2. Any text copied to a buffer shall be in character mode.

113977 If not used as a motion command:

113978 *Current line:* Unchanged.

113979 *Current column:* Set to (*column* - the number of columns occupied by *count* characters ending
113980 with the previous current column).

113981 **Move Down**

113982 *Synopsis:* [count] <newline>
 113983 [count] <control>-J
 113984 [count] <control>-M
 113985 [count] <control>-N
 113986 [count] j
 113987 [count] <carriage-return>
 113988 [count] +

113989 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

113990 If used as a motion command:

- 113991 1. The text region shall include the starting line and the next *count* – 1 lines.
- 113992 2. Any text copied to a buffer shall be in line mode.

113993 If not used as a motion command:

113994 *Current line:* Set to *current line*+ *count*.

113995 *Current column:* Set to non-<blank> for the <carriage-return>, <control>-M, and + commands;
 113996 otherwise, unchanged.

113997 **Clear and Redisplay**

113998 *Synopsis:* <control>-L

113999 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay
 114000 the screen.

114001 *Current line:* Unchanged.

114002 *Current column:* Unchanged.

114003 **Move Up**

114004 *Synopsis:* [count] <control>-P
 114005 [count] k
 114006 [count] –

114007 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

114008 If used as a motion command:

- 114009 1. The text region shall include the starting line and the previous *count* lines.
- 114010 2. Any text copied to a buffer shall be in line mode.

114011 If not used as a motion command:

114012 *Current line:* Set to *current line* – *count*.

114013 *Current column:* Set to non-<blank> for the – command; otherwise, unchanged.

114014 **Redraw Screen**

114015 *Synopsis:* <control>-R

114016 If any lines have been deleted from the display screen and flagged as deleted on the terminal
114017 using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall
114018 be redisplayed to match the contents of the edit buffer.

114019 It is unspecified whether lines flagged with @ because they do not fit on the terminal display
114020 shall be affected.

114021 *Current line:* Unchanged.

114022 *Current column:* Unchanged.

114023 **Scroll Backward**

114024 *Synopsis:* [*count*] <control>-U

114025 If the current line is the first line of the edit buffer, it shall be an error.

114026 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D
114027 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*
114028 shall default to the value of the **scroll** edit option.

114029 *Current line:* If *count* is greater than the current line, set to 1; otherwise, set to the current line -
114030 *count*.

114031 *Current column:* Set to non-<blank>.

114032 **Scroll Backward by Line**

114033 *Synopsis:* [*count*] <control>-Y

114034 Display the line *count* lines before the first line currently displayed.

114035 If the current line is the first line of the edit buffer, it shall be an error. If this calculation would
114036 result in a line that is before the first line of the edit buffer, the first line of the display shall
114037 display some portion of the first line of the edit buffer.

114038 *Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first
114039 line displayed.

114040 *Current column:* Unchanged.

114041 **Edit the Alternate File**

114042 *Synopsis:* <control>-^

114043 This command shall be equivalent to the *ex edit* command, with the alternate pathname as its
114044 argument.

114045 **Terminate Command or Input Mode**114046 *Synopsis:* <ESC>114047 If a partial *vi* command (as defined by at least one, non-*count* character) has been entered,
114048 discard the *count* and the command character(s).114049 Otherwise, if no command characters have been entered, and the <ESC> was the result of a map
114050 expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall
114051 not be an error.

114052 Otherwise, it shall be an error.

114053 *Current line:* Unchanged.114054 *Current column:* Unchanged.114055 **Search for tagstring**114056 *Synopsis:* <control>-]

114057 If the current character is not a word or <blank>, it shall be an error.

114058 This command shall be equivalent to the *ex tag* command, with the argument to that command
114059 defined as follows.

114060 If the current character is a <blank>:

- 114061 1. Skip all <blank> characters after the cursor up to the end of the line.
- 114062 2. If the end of the line is reached, it shall be an error.

114063 Then, the argument to the *ex tag* command shall be the current character and all subsequent
114064 characters, up to the first non-word character or the end of the line.114065 **Move Cursor Forward**114066 *Synopsis:* [*count*] <space>
114067 [*count*] 1 (ell)114068 If there are less than *count* non-<newline> characters after the cursor on the current line, *count*
114069 shall be adjusted to the number of non-<newline> characters after the cursor on the line.

114070 If used as a motion command:

- 114071 1. If the current or *count*th character after the cursor is the last non-<newline> in the line, the
114072 text region shall be comprised of the current character up to and including the last
114073 non-<newline> in the line. Otherwise, the text region shall be from the current character
114074 up to, but not including, the *count*th character after the cursor.
- 114075 2. Any text copied to a buffer shall be in character mode.

114076 If not used as a motion command:

114077 If there are no non-<newline> characters after the current character on the current line, it shall be
114078 an error.114079 *Current line:* Unchanged.114080 *Current column:* Set to the last column that displays any portion of the *count*th character after the
114081 current character.

114082 **Replace Text with Results from Shell Command**

114083 *Synopsis:* [count] ! motion shell-commands <newline>

114084 If the motion command is the ! command repeated:

- 114085 1. If the edit buffer is empty and no *count* was supplied, the command shall be the
114086 equivalent of the *ex* :read ! command, with the text input, and no text shall be copied to
114087 any buffer.
- 114088 2. Otherwise:
 - 114089 a. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be
114090 an error.
 - 114091 b. The text region shall be from the current line up to and including the next *count* -1
114092 lines.

114093 Otherwise, the text region shall be the lines in which any character of the text region specified by
114094 the motion command appear.

114095 Any text copied to a buffer shall be in line mode.

114096 This command shall be equivalent to the *ex* ! command for the specified lines.

114097 **Move Cursor to End-of-Line**

114098 *Synopsis:* [count] \$

114099 It shall be an error if there are less than (*count* -1) lines after the current line in the edit buffer.

114100 If used as a motion command:

- 114101 1. If *count* is 1:
 - 114102 a. It shall be an error if the line is empty.
 - 114103 b. Otherwise, the text region shall consist of all characters from the starting cursor to
114104 the last non-<newline> in the line, inclusive, and any text copied to a buffer shall
114105 be in character mode.
- 114106 2. Otherwise, if the starting cursor position is at or before the first non-<blank> in the line,
114107 the text region shall consist of the current and the next *count* -1 lines, and any text saved
114108 to a buffer shall be in line mode.
- 114109 3. Otherwise, the text region shall consist of all characters from the starting cursor to the last
114110 non-<newline> in the line that is *count* -1 lines forward from the current line, and any text
114111 copied to a buffer shall be in character mode.

114112 If not used as a motion command:

114113 *Current line:* Set to the *current line* + *count*-1.

114114 *Current column:* The current column is set to the last display line column of the last
114115 non-<newline> in the line, or column position 1 if the line is empty.

114116 The current column shall be adjusted to be on the last display line column of the last
114117 non-<newline> of the current line as subsequent commands change the current line, until a
114118 command changes the current column.

114119 **Move to Matching Character**

114120 *Synopsis:* %

114121 If the character at the current position is not a parenthesis, bracket, or curly brace, search
114122 forward in the line to the first one of those characters. If no such character is found, it shall be an
114123 error.

114124 The matching character shall be the parenthesis, bracket, or curly brace matching the
114125 parenthesis, bracket, or curly brace, respectively, that was at the current position or that was
114126 found on the current line.

114127 Matching shall be determined as follows, for an open parenthesis:

- 114128 1. Set a counter to 1.
- 114129 2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.
- 114130 3. If the end of the edit buffer is reached, it shall be an error.
- 114131 4. If an open parenthesis is found, increment the counter by 1.
- 114132 5. If a close parenthesis is found, decrement the counter by 1.
- 114133 6. If the counter is zero, the current character is the matching character.

114134 Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,
114135 from the starting character to the beginning of the buffer, a close parenthesis shall increment the
114136 counter by 1, and an open parenthesis shall decrement the counter by 1.

114137 Matching for brackets and curly braces shall be equivalent, except that searching shall be done
114138 for open and close brackets or open and close curly braces. It is implementation-defined whether
114139 other characters are searched for and matched as well.

114140 If used as a motion command:

- 114141 1. If the matching cursor was after the starting cursor in the edit buffer, and the starting
114142 cursor position was at or before the first non-<blank> non-<newline> in the starting line,
114143 and the matching cursor position was at or after the last non-<blank> non-<newline> in
114144 the matching line, the text region shall consist of the current line to the matching line,
114145 inclusive, and any text copied to a buffer shall be in line mode.
- 114146 2. If the matching cursor was before the starting cursor in the edit buffer, and the starting
114147 cursor position was at or after the last non-<blank> non-<newline> in the starting line,
114148 and the matching cursor position was at or before the first non-<blank> non-<newline> in
114149 the matching line, the text region shall consist of the current line to the matching line,
114150 inclusive, and any text copied to a buffer shall be in line mode.
- 114151 3. Otherwise, the text region shall consist of the starting character to the matching character,
114152 inclusive, and any text copied to a buffer shall be in character mode.

114153 If not used as a motion command:

114154 *Current line:* Set to the line where the matching character is located.

114155 *Current column:* Set to the last column where any portion of the matching character is displayed.

114156 **Repeat Substitution**

114157 *Synopsis:* &

114158 Repeat the previous substitution command. This command shall be equivalent to the *ex &*
114159 command with the current line as its addresses, and without *options, count, or flags*.

114160 **Return to Previous Context at Beginning of Line**

114161 *Synopsis:* ' *character*

114162 It shall be an error if there is no line in the edit buffer marked by *character*.

114163 If used as a motion command:

- 114164 1. If the starting cursor is after the marked cursor, then the locations of the starting cursor
114165 and the marked cursor in the edit buffer shall be logically swapped.
- 114166 2. The text region shall consist of the starting line up to and including the marked line, and
114167 any text copied to a buffer shall be in line mode.

114168 If not used as a motion command:

114169 *Current line:* Set to the line referenced by the mark.

114170 *Current column:* Set to non-<blank>.

114171 **Return to Previous Context**

114172 *Synopsis:* ` *character*

114173 It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer
114174 contains a character in the saved numbered character position, it shall be as if the marked
114175 position is the first non-<blank>.

114176 If used as a motion command:

- 114177 1. It shall be an error if the marked cursor references the same character in the edit buffer as
114178 the starting cursor.
- 114179 2. If the starting cursor is after the marked cursor, then the locations of the starting cursor
114180 and the marked cursor in the edit buffer shall be logically swapped.
- 114181 3. If the starting line is empty or the starting cursor is at or before the first non-<blank>
114182 non-<newline> of the starting line, and the marked cursor line is empty or the marked
114183 cursor references the first character of the marked cursor line, the text region shall consist
114184 of all lines containing characters from the starting cursor to the line before the marked
114185 cursor line, inclusive, and any text copied to a buffer shall be in line mode.
- 114186 4. Otherwise, if the marked cursor line is empty or the marked cursor references a character
114187 at or before the first non-<blank> non-<newline> of the marked cursor line, the region of
114188 text shall be from the starting cursor to the last non-<newline> of the line before the
114189 marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.
- 114190 5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked
114191 cursor (exclusive), and any text copied to a buffer shall be in character mode.

114192 If not used as a motion command:

114193 *Current line:* Set to the line referenced by the mark.

114194 *Current column:* Set to the last column in which any portion of the character referenced by the

114195 mark is displayed.

114196 **Return to Previous Section**

114197 *Synopsis:* [*count*] [[

114198 Move the cursor backward through the edit buffer to the first character of the previous section
114199 boundary, *count* times.

114200 If used as a motion command:

- 114201 1. If the starting cursor was at the first character of the starting line or the starting line was
114202 empty, and the first character of the boundary was the first character of the boundary line,
114203 the text region shall consist of the current line up to and including the line where the
114204 *count*th next boundary starts, and any text copied to a buffer shall be in line mode.
- 114205 2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last
114206 line of the edit buffer, the text region shall consist of the last character in the edit buffer up
114207 to and including the starting character, and any text saved to a buffer shall be in character
114208 mode.
- 114209 3. Otherwise, the text region shall consist of the starting character up to but not including
114210 the first character in the *count*th next boundary, and any text copied to a buffer shall be in
114211 character mode.

114212 If not used as a motion command:

114213 *Current line:* Set to the line where the *count*th next boundary in the edit buffer starts.

114214 *Current column:* Set to the last column in which any portion of the first character of the *count*th
114215 next boundary is displayed, or column position 1 if the line is empty.

114216 **Move to Next Section**

114217 *Synopsis:* [*count*]]]

114218 Move the cursor forward through the edit buffer to the first character of the next section
114219 boundary, *count* times.

114220 If used as a motion command:

- 114221 1. If the starting cursor was at the first character of the starting line or the starting line was
114222 empty, and the first character of the boundary was the first character of the boundary line,
114223 the text region shall consist of the current line up to and including the line where the
114224 *count*th previous boundary starts, and any text copied to a buffer shall be in line mode.
- 114225 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first
114226 character in the edit buffer up to but not including the starting character, and any text
114227 copied to a buffer shall be in character mode.
- 114228 3. Otherwise, the text region shall consist of the first character in the *count*th previous
114229 section boundary up to but not including the starting character, and any text copied to a
114230 buffer shall be in character mode.

114231 If not used as a motion command:

114232 *Current line:* Set to the line where the *count*th previous boundary in the edit buffer starts.

114233 *Current column:* Set to the last column in which any portion of the first character of the *count*th
114234 previous boundary is displayed, or column position 1 if the line is empty.

114235 **Move to First Non-<blank> Position on Current Line**

114236 *Synopsis:* `^`

114237 If used as a motion command:

- 114238 1. If the line has no non-<blank> non-<newline> characters, or if the cursor is at the first
114239 non-<blank> non-<newline> of the line, it shall be an error.
- 114240 2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region
114241 shall be comprised of the current character, up to, but not including, the first non-<blank>
114242 non-<newline> of the line.
- 114243 3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall
114244 be from the character before the starting cursor up to and including the first non-<blank>
114245 non-<newline> of the line.
- 114246 4. Any text copied to a buffer shall be in character mode.

114247 If not used as a motion command:

114248 *Current line:* Unchanged.

114249 *Current column:* Set to non-<blank>.

114250 **Current and Line Above**

114251 *Synopsis:* `[count] _`

114252 If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an error.

114253 If used as a motion command:

- 114254 1. If *count* is less than 2, the text region shall be the current line.
- 114255 2. Otherwise, the text region shall include the starting line and the next *count* -1 lines.
- 114256 3. Any text copied to a buffer shall be in line mode.

114257 If not used as a motion command:

114258 *Current line:* Set to current line + *count* -1.

114259 *Current column:* Set to non-<blank>.

114260 **Move Back to Beginning of Sentence**

114261 *Synopsis:* `[count] (`

114262 Move backward to the beginning of a sentence. This command shall be equivalent to the `[[`
114263 `command`, with the exception that sentence boundaries shall be used instead of section
114264 boundaries.

114265 **Move Forward to Beginning of Sentence**

114266 *Synopsis:* `[count])`

114267 Move forward to the beginning of a sentence. This command shall be equivalent to the `]]`
114268 `command`, with the exception that sentence boundaries shall be used instead of section
114269 boundaries.

114270 **Move Back to Preceding Paragraph**

114271 *Synopsis:* [*count*] {

114272 Move back to the beginning of the preceding paragraph. This command shall be equivalent to
114273 the [[command, with the exception that paragraph boundaries shall be used instead of section
114274 boundaries.

114275 **Move Forward to Next Paragraph**

114276 *Synopsis:* [*count*] }

114277 Move forward to the beginning of the next paragraph. This command shall be equivalent to the
114278]] command, with the exception that paragraph boundaries shall be used instead of section
114279 boundaries.

114280 **Move to Specific Column Position**

114281 *Synopsis:* [*count*] |

114282 For the purposes of this command, lines that are too long for the current display and that have
114283 been folded shall be treated as having a single, 1-based, number of columns.

114284 If there are less than *count* columns in which characters from the current line are displayed on
114285 the screen, *count* shall be adjusted to be the last column in which any portion of the line is
114286 displayed on the screen.

114287 If used as a motion command:

- 114288 1. If the line is empty, or the cursor character is the same as the character on the *count*th
114289 column of the line, it shall be an error.
- 114290 2. If the cursor is before the *count*th column of the line, the text region shall be comprised of
114291 the current character, up to but not including the character on the *count*th column of the
114292 line.
- 114293 3. If the cursor is after the *count*th column of the line, the text region shall be from the
114294 character before the starting cursor up to and including the character on the *count*th
114295 column of the line.
- 114296 4. Any text copied to a buffer shall be in character mode.

114297 If not used as a motion command:

114298 *Current line:* Unchanged.

114299 *Current column:* Set to the last column in which any portion of the character that is displayed in
114300 the *count* column of the line is displayed.

114301 **Reverse Find Character**

114302 *Synopsis:* [*count*] ,

114303 If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or
114304 **T** command, respectively, with the specified *count* and the same search character.

114305 If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

114306 Repeat

114307 *Synopsis:* [count] .

114308 Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall
 114309 be an error if none of these commands have been executed. Commands (other than commands
 114310 that enter text input mode) executed as a result of map expansions, shall not change the value of
 114311 the last repeatable command.

114312 Repeated commands with associated motion commands shall repeat the motion command as
 114313 well; however, any specified *count* shall replace the *count*(s) that were originally specified to the
 114314 repeated command or its associated motion command.

114315 If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall
 114316 not set the remembered search character for the **;** and **,** commands.

114317 If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric
 114318 buffer named with a number less than 9, the buffer associated with the repeated command shall
 114319 be set to be the buffer named by the name of the previous buffer logically incremented by 1.

114320 If the repeated character is a text input command, the input text associated with that command
 114321 is repeated literally:

114322 Input characters are neither macro or abbreviation-expanded.

114323 Input characters are not interpreted in any special way with the exception that <newline>,
 114324 <carriage-return>, and <control>-T behave as described in [Input Mode Commands in vi](#)
 114325 (on page 3410).

114326 *Current line:* Set as described for the repeated command.

114327 *Current column:* Set as described for the repeated command.

114328 Find Regular Expression

114329 *Synopsis:* /

114330 If the input line contains no non-<newline> characters, it shall be equivalent to a line containing
 114331 only the last regular expression encountered. The enhanced regular expressions supported by *vi*
 114332 are described in [Regular Expressions in ex](#) (on page 2734).

114333 Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed
 114334 by an address offset or a *vi z* command.

114335 If the regular expression is not the last regular expression on the line, or if a line offset or **z**
 114336 command is specified, the regular expression shall be terminated by an unescaped **'/'**
 114337 character, which shall not be used as part of the regular expression. If the regular expression is
 114338 not the first regular expression on the line, it shall be preceded by zero or more <blank>
 114339 characters, a <semicolon>, zero or more <blank> characters, and a leading **'/'** character, which
 114340 shall not be interpreted as part of the regular expression. It shall be an error to precede any
 114341 regular expression with any characters other than these.

114342 Each search shall begin from the character after the first character of the last match (or, if it is the
 114343 first search, after the cursor). If the **wraps**can edit option is set, the search shall continue to the
 114344 character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be
 114345 an error if any search fails to find a match, and an informational message to this effect shall be
 114346 displayed.

114347 An optional address offset (see [Addressing in ex](#), on page 2703) can be specified after the last
 114348 regular expression by including a trailing **'/'** character after the regular expression and

114349 specifying the address offset. This offset will be from the line containing the match for the last
 114350 regular expression specified. It shall be an error if the line offset would indicate a line address
 114351 less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be
 114352 supported. It shall be an error to follow the address offset with any other characters than
 114353 <blank> characters.

114354 If not used as a motion command, an optional **z** command (see [Redraw Window](#), on page 3409)
 114355 can be specified after the last regular expression by including a trailing '/' character after the
 114356 regular expression, zero or more <blank> characters, a 'z', zero or more <blank> characters, an
 114357 optional new **window** edit option value, zero or more <blank> characters, and a location
 114358 character. The effect shall be as if the **z** command was executed after the / command. It shall be
 114359 an error to follow the **z** command with any other characters than <blank> characters.

114360 The remembered search direction shall be set to forward.

114361 If used as a motion command:

- 114362 1. It shall be an error if the last match references the same character in the edit buffer as the
 114363 starting cursor.
- 114364 2. If any address offset is specified, the last match shall be adjusted by the specified offset as
 114365 described previously.
- 114366 3. If the starting cursor is after the last match, then the locations of the starting cursor and
 114367 the last match in the edit buffer shall be logically swapped.
- 114368 4. If any address offset is specified, the text region shall consist of all lines containing
 114369 characters from the starting cursor to the last match line, inclusive, and any text copied to
 114370 a buffer shall be in line mode.
- 114371 5. Otherwise, if the starting line is empty or the starting cursor is at or before the first
 114372 non-<blank> non-<newline> of the starting line, and the last match line is empty or the
 114373 last match starts at the first character of the last match line, the text region shall consist of
 114374 all lines containing characters from the starting cursor to the line before the last match
 114375 line, inclusive, and any text copied to a buffer shall be in line mode.
- 114376 6. Otherwise, if the last match line is empty or the last match begins at a character at or
 114377 before the first non-<blank> non-<newline> of the last match line, the region of text shall
 114378 be from the current cursor to the last non-<newline> of the line before the last match line,
 114379 inclusive, and any text copied to a buffer shall be in character mode.
- 114380 7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first
 114381 character of the last match (exclusive), and any text copied to a buffer shall be in character
 114382 mode.

114383 If not used as a motion command:

114384 *Current line*: If a match is found, set to the last matched line plus the address offset, if any;
 114385 otherwise, unchanged.

114386 *Current column*: Set to the last column on which any portion of the first character in the last
 114387 matched string is displayed, if a match is found; otherwise, unchanged.

114388 **Move to First Character in Line**

114389 *Synopsis:* 0 (zero)

114390 Move to the first character on the current line. The character '0' shall not be interpreted as a
114391 command if it is immediately preceded by a digit.

114392 If used as a motion command:

- 114393 1. If the cursor character is the first character in the line, it shall be an error.
- 114394 2. The text region shall be from the character before the cursor character up to and including
114395 the first character in the line.
- 114396 3. Any text copied to a buffer shall be in character mode.

114397 If not used as a motion command:

114398 *Current line:* Unchanged.

114399 *Current column:* The last column in which any portion of the first character in the line is
114400 displayed, or if the line is empty, unchanged.

114401 **Execute an ex Command**

114402 *Synopsis:* :

114403 Execute one or more *ex* commands.

114404 If any portion of the screen other than the last line of the screen was overwritten by any *ex*
114405 command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from
114406 the user, and shall then read a character. This action may also be taken for other, unspecified
114407 reasons.

114408 If the next character entered is a ':', another *ex* command shall be accepted and executed. Any
114409 other character shall cause the screen to be refreshed and *vi* shall return to command mode.

114410 *Current line:* As specified for the *ex* command.

114411 *Current column:* As specified for the *ex* command.

114412 **Repeat Find**

114413 *Synopsis:* [*count*] ;

114414 This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and
114415 with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**,
114416 **f**, **T**, or **t** command, it shall be an error.

114417 **Shift Left**

114418 *Synopsis:* [*count*] < *motion*

114419 If the motion command is the < command repeated:

- 114420 1. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an
114421 error.
- 114422 2. The text region shall be from the current line, up to and including the next *count* - 1 lines.

114423 Shift any line in the text region specified by the *count* and motion command one shiftwidth (see
114424 the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The
114425 unshifted lines shall be copied to the unnamed buffer in line mode.

114426 *Current line*: If the motion was from the current cursor position toward the end of the edit buffer,
 114427 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region
 114428 specified by the motion command.

114429 *Current column*: Set to non-<blank>.

114430 **Shift Right**

114431 *Synopsis*: [count] > motion

114432 If the motion command is the > command repeated:

- 114433 1. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an
 114434 error.
- 114435 2. The text region shall be from the current line, up to and including the next *count* - 1 lines.

114436 Shift any line with characters in the text region specified by the *count* and motion command one
 114437 shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* >
 114438 command. The unshifted lines shall be copied into the unnamed buffer in line mode.

114439 *Current line*: If the motion was from the current cursor position toward the end of the edit buffer,
 114440 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region
 114441 specified by the motion command.

114442 *Current column*: Set to non-<blank>.

114443 **Scan Backwards for Regular Expression**

114444 *Synopsis*: ?

114445 Scan backwards; the ? command shall be equivalent to the / command (see [Find Regular](#)
 114446 [Expression](#), on page 3392) with the following exceptions:

- 114447 1. The input prompt shall be a ' ? '.
- 114448 2. Each search shall begin from the character before the first character of the last match (or, if
 114449 it is the first search, the character before the cursor character).
- 114450 3. The search direction shall be from the cursor toward the beginning of the edit buffer, and
 114451 the **wrapscan** edit option shall affect whether the search wraps to the end of the edit
 114452 buffer and continues.
- 114453 4. The remembered search direction shall be set to backward.

114454 **Execute**

114455 *Synopsis*: @buffer

114456 If the *buffer* is specified as @, the last buffer executed shall be used. If no previous buffer has been
 114457 executed, it shall be an error.

114458 Behave as if the contents of the named buffer were entered as standard input. After each line of a
 114459 line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline>
 114460 were entered as standard input.

114461 If an error occurs during this process, an error message shall be written, and no more characters
 114462 resulting from the execution of this command shall be processed.

114463 If a *count* is specified, behave as if that count were entered as user input before the characters
 114464 from the @ buffer were entered.

114465 *Current line*: As specified for the individual commands.

114466 *Current column*: As specified for the individual commands.

114467 **Reverse Case**

114468 *Synopsis*: [count] ~

114469 Reverse the case of the current character and the next *count* -1 characters, such that lowercase
114470 characters that have uppercase counterparts shall be changed to uppercase characters, and
114471 uppercase characters that have lowercase counterparts shall be changed to lowercase characters,
114472 as prescribed by the current locale. No other characters shall be affected by this command.

114473 If there are less than *count* -1 characters after the cursor in the edit buffer, *count* shall be adjusted
114474 to the number of characters after the cursor in the edit buffer minus 1.

114475 For the purposes of this command, the next character after the last non-<newline> on the line
114476 shall be the next character in the edit buffer.

114477 *Current line*: Set to the line including the (*count*-1)th character after the cursor.

114478 *Current column*: Set to the last column in which any portion of the (*count*-1)th character after the
114479 cursor is displayed.

114480 **Append**

114481 *Synopsis*: [count] a

114482 Enter text input mode after the current cursor position. No characters already in the edit buffer
114483 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1
114484 more times to the end of the input.

114485 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),
114486 on page 3410).

114487 **Append at End-of-Line**

114488 *Synopsis*: [count] A

114489 This command shall be equivalent to the *vi* command:

114490 \$ [count] a

114491 (see [Append](#)).

114492 **Move Backward to Preceding Word**

114493 *Synopsis*: [count] b

114494 With the exception that words are used as the delimiter instead of bigwords, this command shall
114495 be equivalent to the **B** command.

114496 **Move Backward to Preceding Bigword**114497 *Synopsis:* [count] B

114498 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an
 114499 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*
 114500 shall be adjusted to the number of bigword beginnings between the cursor and the start of the
 114501 edit buffer.

114502 If used as a motion command:

- 114503 1. The text region shall be from the first character of the *count*th previous bigword beginning
 114504 up to but not including the cursor character.
- 114505 2. Any text copied to a buffer shall be in character mode.

114506 If not used as a motion command:

114507 *Current line:* Set to the line containing the *current column*.

114508 *Current column:* Set to the last column upon which any part of the first character of the *count*th
 114509 previous bigword is displayed.

114510 **Change**114511 *Synopsis:* [buffer] [count] c motion114512 If the motion command is the **c** command repeated:

- 114513 1. The buffer text shall be in line mode.
- 114514 2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an
 114515 error.
- 114516 3. The text region shall be from the current line up to and including the next *count* -1 lines.

114517 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

114518 The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text
 114519 to be replaced contains characters from more than a single line, or the buffer text is in line mode,
 114520 the replaced text shall be copied into the numeric buffers as well.

114521 If the buffer text is in line mode:

- 114522 1. Any lines that contain characters in the region shall be deleted, and the editor shall enter
 114523 text input mode at the beginning of a new line which shall replace the first line deleted.
- 114524 2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**
 114525 characters on the first line deleted shall be inserted as if entered by the user.

114526 Otherwise, if characters from more than one line are in the region of text:

- 114527 1. The text shall be deleted.
- 114528 2. Any text remaining in the last line in the text region shall be appended to the first line in
 114529 the region, and the last line in the region shall be deleted.
- 114530 3. The editor shall enter text input mode after the last character not deleted from the first
 114531 line in the text region, if any; otherwise, on the first column of the first line in the region.

114532 Otherwise:

114533 1. If the glyph for '\$' is smaller than the region, the end of the region shall be marked with
114534 a '\$'.

114535 2. The editor shall enter text input mode, overwriting the region of text.

114536 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),
114537 on page 3410).

114538 **Change to End-of-Line**

114539 *Synopsis:* `[buffer] [count] C`

114540 This command shall be equivalent to the *vi* command:

114541 `[buffer] [count] c$`

114542 See the **c** command.

114543 **Delete**

114544 *Synopsis:* `[buffer] [count] d motion`

114545 If the motion command is the **d** command repeated:

114546 1. The buffer text shall be in line mode.

114547 2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an
114548 error.

114549 3. The text region shall be from the current line up to and including the next *count* -1 lines.

114550 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

114551 If in open mode, and the current line is deleted, and the line remains on the display, an '@'
114552 character shall be displayed as the first glyph of that line.

114553 Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be
114554 deleted contains characters from more than a single line, or the buffer text is in line mode, the
114555 deleted text shall be copied into the numeric buffers, as well.

114556 *Current line:* Set to the first text region line that appears in the edit buffer, unless that line has
114557 been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit
114558 buffer is empty.

114559 *Current column:*

114560 1. If the line is empty, set to column position 1.

114561 2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the
114562 end of the edit buffer:

114563 a. If a character from the current line is displayed in the current column, set to the
114564 last column that displays any portion of that character.

114565 b. Otherwise, set to the last column in which any portion of any character in the line
114566 is displayed.

114567 3. Otherwise, if a character is displayed in the column that began the text region, set to the
114568 last column that displays any portion of that character.

114569 4. Otherwise, set to the last column in which any portion of any character in the line is
114570 displayed.

114571 **Delete to End-of-Line**

114572 *Synopsis:* [buffer] D

114573 Delete the text from the current position to the end of the current line; equivalent to the *vi*
114574 command:

114575 [buffer] d\$

114576 **Move to End-of-Word**

114577 *Synopsis:* [count] e

114578 With the exception that words are used instead of bigwords as the delimiter, this command shall
114579 be equivalent to the **E** command.

114580 **Move to End-of-Bigword**

114581 *Synopsis:* [count] E

114582 If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor
114583 and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between
114584 the cursor and the end of the edit buffer.

114585 If used as a motion command:

- 114586 1. The text region shall be from the last character of the *count*th next bigword up to and
114587 including the cursor character.
- 114588 2. Any text copied to a buffer shall be in character mode.

114589 If not used as a motion command:

114590 *Current line:* Set to the line containing the current column.

114591 *Current column:* Set to the last column upon which any part of the last character of the *count*th
114592 next bigword is displayed.

114593 **Find Character in Current Line (Forward)**

114594 *Synopsis:* [count] f character

114595 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

114596 If used as a motion command:

- 114597 1. The text range shall be from the cursor character up to and including the *count*th
114598 occurrence of the specified character after the cursor.
- 114599 2. Any text copied to a buffer shall be in character mode.

114600 If not used as a motion command:

114601 *Current line:* Unchanged.

114602 *Current column:* Set to the last column in which any portion of the *count*th occurrence of the
114603 specified character after the cursor appears in the line.

114604 **Find Character in Current Line (Reverse)**

114605 *Synopsis:* [count] F character

114606 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

114607 If used as a motion command:

- 114608 1. The text region shall be from the *count*th occurrence of the specified character before the
- 114609 cursor, up to, but not including the cursor character.
- 114610 2. Any text copied to a buffer shall be in character mode.

114611 If not used as a motion command:

114612 *Current line:* Unchanged.

114613 *Current column:* Set to the last column in which any portion of the *count*th occurrence of the

114614 specified character before the cursor appears in the line.

114615 **Move to Line**

114616 *Synopsis:* [count] G

114617 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than

114618 the last line of the edit buffer, it shall be an error.

114619 If used as a motion command:

- 114620 1. The text region shall be from the cursor line up to and including the specified line.
- 114621 2. Any text copied to a buffer shall be in line mode.

114622 If not used as a motion command:

114623 *Current line:* Set to *count* if *count* is specified; otherwise, the last line.

114624 *Current column:* Set to non-<blank>.

114625 **Move to Top of Screen**

114626 *Synopsis:* [count] H

114627 If the beginning of the line *count* greater than the first line of which any portion appears on the

114628 display does not exist, it shall be an error.

114629 If used as a motion command:

- 114630 1. If in open mode, the text region shall be the current line.
- 114631 2. Otherwise, the text region shall be from the starting line up to and including (the first line
- 114632 of the display + *count* -1).
- 114633 3. Any text copied to a buffer shall be in line mode.

114634 If not used as a motion command:

114635 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

114636 Otherwise, it shall set the current line and current column as follows.

114637 *Current line:* Set to (the first line of the display + *count* -1).

114638 *Current column:* Set to non-<blank>.

114639 **Insert Before Cursor**114640 *Synopsis:* [count] i

114641 Enter text input mode before the current cursor position. No characters already in the edit buffer
 114642 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1
 114643 more times to the end of the input.

114644 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),
 114645 on page 3410).

114646 **Insert at Beginning of Line**114647 *Synopsis:* [count] I114648 This command shall be equivalent to the *vi* command \wedge [count]i.114649 **Join**114650 *Synopsis:* [count] J

114651 If the current line is the last line in the edit buffer, it shall be an error.

114652 This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex*
 114653 command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex*
 114654 command *count* value of *count* -1 for any other value of *count*, except that the current line and
 114655 column shall be set as follows.

114656 *Current line:* Unchanged.

114657 *Current column:* The last column in which any portion of the character following the last
 114658 character in the initial line is displayed, or the last non-<newline> in the line if no characters
 114659 were appended.

114660 **Move to Bottom of Screen**114661 *Synopsis:* [count] L

114662 If the beginning of the line *count* less than the last line of which any portion appears on the
 114663 display does not exist, it shall be an error.

114664 If used as a motion command:

- 114665 1. If in open mode, the text region shall be the current line.
- 114666 2. Otherwise, the text region shall include all lines from the starting cursor line to (the last
114667 line of the display -(*count* -1)).
- 114668 3. Any text copied to a buffer shall be in line mode.

114669 If not used as a motion command:

- 114670 1. If in open mode, this command shall set the current column to non-<blank> and do
114671 nothing else.
- 114672 2. Otherwise, it shall set the current line and current column as follows.

114673 *Current line:* Set to (the last line of the display -(*count* -1)).114674 *Current column:* Set to non-<blank>.

114675 **Mark Position**

114676 *Synopsis:* m *letter*

114677 This command shall be equivalent to the *ex mark* command with the specified character as an
114678 argument.

114679 **Move to Middle of Screen**

114680 *Synopsis:* M

114681 The middle line of the display shall be calculated as follows:

114682 (the top line of the display) + (((number of lines displayed) + 1) / 2) - 1

114683 If used as a motion command:

- 114684 1. If in open mode, the text region shall be the current line.
- 114685 2. Otherwise, the text region shall include all lines from the starting cursor line up to and
114686 including the middle line of the display.
- 114687 3. Any text copied to a buffer shall be in line mode.

114688 If not used as a motion command:

114689 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

114690 Otherwise, it shall set the current line and current column as follows.

114691 *Current line:* Set to the middle line of the display.

114692 *Current column:* Set to non-<blank>.

114693 **Repeat Regular Expression Find (Forward)**

114694 *Synopsis:* n

114695 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi /*
114696 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi ?*
114697 command with no characters entered by the user.

114698 If the **n** command is used as a motion command for the **!** command, the editor shall not enter
114699 text input mode on the last line on the screen, and shall behave as if the user entered a single
114700 '!' character as the text input.

114701 **Repeat Regular Expression Find (Reverse)**

114702 *Synopsis:* N

114703 Scan for the next match of the last pattern given to */* or *?*, but in the reverse direction; this is the
114704 reverse of **n**.

114705 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi ?*
114706 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi /*
114707 command with no characters entered by the user. If the **N** command is used as a motion
114708 command for the **!** command, the editor shall not enter text input mode on the last line on the
114709 screen, and shall behave as if the user entered a single **!** character as the text input.

114710 **Insert Empty Line Below**114711 *Synopsis:* ○

114712 Enter text input mode in a new line appended after the current line. A *count* shall cause the input
 114713 text to be appended *count* -1 more times to the end of the already added text, each time starting
 114714 on a new, appended line.

114715 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),
 114716 on page 3410).

114717 **Insert Empty Line Above**114718 *Synopsis:* ○

114719 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input
 114720 text to be appended *count* -1 more times to the end of the already added text, each time starting
 114721 on a new, appended line.

114722 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),
 114723 on page 3410).

114724 **Put from Buffer Following**114725 *Synopsis:* [*buffer*] p114726 If no *buffer* is specified, the unnamed buffer shall be used.

114727 If the buffer text is in line mode, the text shall be appended below the current line, and each line
 114728 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be
 114729 appended *count* -1 more times to the end of the already added text, each time starting on a new,
 114730 appended line.

114731 If the buffer text is in character mode, the text shall be appended into the current line after the
 114732 cursor, and each line of the buffer other than the first and last shall become a new line in the edit
 114733 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the
 114734 already added text, each time starting after the last added character.

114735 *Current line:* If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.

114736 *Current column:* If the buffer text is in line mode:

- 114737 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any
 114738 portion of the first non-<blank> in the line is displayed.
- 114739 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which
 114740 any portion of the last non-<newline> in the first line of the buffer is displayed.

114741 If the buffer text is in character mode:

- 114742 1. If the text in the buffer is from more than a single line, then set to the last column on
 114743 which any portion of the first character from the buffer is displayed.
- 114744 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any
 114745 portion of the last character from the buffer is displayed.
- 114746 3. Otherwise, set to the first column on which any portion of the first character from the
 114747 buffer is displayed.

114748 **Put from Buffer Before**

114749 *Synopsis:* `[buffer] P`

114750 If no *buffer* is specified, the unnamed buffer shall be used.

114751 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of
114752 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be
114753 appended *count* -1 more times to the end of the already added text, each time starting on a new,
114754 appended line.

114755 If the buffer text is in character mode, the text shall be inserted into the current line before the
114756 cursor, and each line of the buffer other than the first and last shall become a new line in the edit
114757 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the
114758 already added text, each time starting after the last added character.

114759 *Current line:* Unchanged.

114760 *Current column:* If the buffer text is in line mode:

- 114761 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any
114762 portion of that character is displayed.
- 114763 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which
114764 any portion of the last non-<newline> in the first line of the buffer is displayed.

114765 If the buffer text is in character mode:

- 114766 1. If the text in the buffer is from more than a single line, then set to the last column on
114767 which any portion of the first character from the buffer is displayed.
- 114768 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any
114769 portion of the last character from the buffer is displayed.
- 114770 3. Otherwise, set to the first column on which any portion of the first character from the
114771 buffer is displayed.

114772 **Enter ex Mode**

114773 *Synopsis:* `Q`

114774 Leave visual or open mode and enter *ex* command mode.

114775 *Current line:* Unchanged.

114776 *Current column:* Unchanged.

114777 **Replace Character**

114778 *Synopsis:* `[count] r character`

114779 Replace the *count* characters at and after the cursor with the specified character. If there are less
114780 than *count* non-<newline> characters at and after the cursor on the line, it shall be an error.

114781 If character is <control>-V, any next character other than the <newline> shall be stripped of any
114782 special meaning and used as a literal character.

114783 If character is <ESC>, no replacement shall be made and the current line and current column
114784 shall be unchanged.

114785 If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current
114786 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be

114787 discarded, and any remaining characters after the cursor in the current line shall be moved to the
 114788 last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same
 114789 number of **autoindent** characters found on the line from which the command was executed.

114790 *Current line*: Unchanged unless the replacement character is a <carriage-return> or <newline>, in
 114791 which case it shall be set to line + *count*.

114792 *Current column*: Set to the last column position on which a portion of the last replaced character
 114793 is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.

114794 **Replace Characters**

114795 *Synopsis*: R

114796 Enter text input mode at the current cursor position possibly replacing text on the current line. A
 114797 *count* shall cause the input text to be appended *count* - 1 more times to the end of the input.

114798 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),
 114799 on page 3410).

114800 **Substitute Character**

114801 *Synopsis*: [*buffer*] [*count*] s

114802 This command shall be equivalent to the *vi* command:

114803 [*buffer*] [*count*] c<space>

114804 **Substitute Lines**

114805 *Synopsis*: [*buffer*] [*count*] S

114806 This command shall be equivalent to the *vi* command:

114807 [*buffer*] [*count*] c_

114808 **Move Cursor to Before Character (Forward)**

114809 *Synopsis*: [*count*] t *character*

114810 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

114811 If used as a motion command:

- 114812 1. The text region shall be from the cursor up to but not including the *count*th occurrence of
 114813 the specified character after the cursor.
- 114814 2. Any text copied to a buffer shall be in character mode.

114815 If not used as a motion command:

114816 *Current line*: Unchanged.

114817 *Current column*: Set to the last column in which any portion of the character before the *count*th
 114818 occurrence of the specified character after the cursor appears in the line.

114819 **Move Cursor to After Character (Reverse)**

114820 *Synopsis:* [count] T character

114821 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

114822 If used as a motion command:

- 114823 1. If the character before the cursor is the specified character, it shall be an error.
- 114824 2. The text region shall be from the character before the cursor up to but not including the
- 114825 *count*th occurrence of the specified character before the cursor.
- 114826 3. Any text copied to a buffer shall be in character mode.

114827 If not used as a motion command:

114828 *Current line:* Unchanged.

114829 *Current column:* Set to the last column in which any portion of the character after the *count*th
114830 occurrence of the specified character before the cursor appears in the line.

114831 **Undo**

114832 *Synopsis:* u

114833 This command shall be equivalent to the *ex* **undo** command except that the current line and
114834 current column shall be set as follows:

114835 *Current line:* Set to the first line added or changed if any; otherwise, move to the line preceding
114836 any deleted text if one exists; otherwise, move to line 1.

114837 *Current column:* If undoing an *ex* command, set to the first non-<blank>.

114838 Otherwise, if undoing a text input command:

- 114839 1. If the command was a **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to
114840 the value it held when the text input command was entered.
- 114841 2. Otherwise, set to the last column in which any portion of the first character after the
114842 deleted text is displayed, or, if no non-<newline> characters follow the text deleted from
114843 this line, set to the last column in which any portion of the last non-<newline> in the line
114844 is displayed, or 1 if the line is empty.

114845 Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

- 114846 1. If text was added or changed, set to the last column in which any portion of the first
114847 character added or changed is displayed.
- 114848 2. If text was deleted, set to the last column in which any portion of the first character after
114849 the deleted text is displayed, or, if no non-<newline> characters follow the deleted text,
114850 set to the last column in which any portion of the last non-<newline> in the line is
114851 displayed, or 1 if the line is empty.

114852 Otherwise, set to non-<blank>.

114853 **Undo Current Line**114854 *Synopsis:* U114855 Restore the current line to its state immediately before the most recent time that it became the
114856 current line.114857 *Current line:* Unchanged.114858 *Current column:* Set to the first column in the line in which any portion of the first character in
114859 the line is displayed.114860 **Move to Beginning of Word**114861 *Synopsis:* [count] w114862 With the exception that words are used as the delimiter instead of bigwords, this command shall
114863 be equivalent to the **W** command.114864 **Move to Beginning of Bigword**114865 *Synopsis:* [count] W114866 If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the
114867 cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last
114868 bigword in the edit buffer.

114869 If used as a motion command:

- 114870 1. If the associated command is **c**, *count* is 1, and the cursor is on a <blank>, the region of
114871 text shall be the current character and no further action shall be taken.
- 114872 2. If there are less than *count* bigwords between the cursor and the end of the edit buffer,
114873 then the command shall succeed, and the region of text shall include the last character of
114874 the edit buffer.
- 114875 3. If there are <blank> characters or an end-of-line that precede the *count*th bigword, and the
114876 associated command is **c**, the region of text shall be up to and including the last character
114877 before the preceding <blank> characters or end-of-line.
- 114878 4. If there are <blank> characters or an end-of-line that precede the bigword, and the
114879 associated command is **d** or **y**, the region of text shall be up to and including the last
114880 <blank> before the start of the bigword or end-of-line.
- 114881 5. Any text copied to a buffer shall be in character mode.

114882 If not used as a motion command:

- 114883 1. If the cursor is on the last character of the edit buffer, it shall be an error.

114884 *Current line:* Set to the line containing the current column.114885 *Current column:* Set to the last column in which any part of the first character of the *count*th next
114886 bigword is displayed.

114887 **Delete Character at Cursor**

114888 *Synopsis:* `[buffer] [count] x`

114889 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the
114890 unnamed buffer.

114891 If the line is empty, it shall be an error. If there are less than *count* non-<newline> characters at
114892 and after the cursor on the current line, *count* shall be adjusted to the number of non-<newline>
114893 characters at and after the cursor.

114894 *Current line:* Unchanged.

114895 *Current column:* If the line is empty, set to column position 1. Otherwise, if there were *count* or
114896 less non-<newline> characters at and after the cursor on the current line, set to the last column
114897 that displays any part of the last non-<newline> of the line. Otherwise, unchanged.

114898 **Delete Character Before Cursor**

114899 *Synopsis:* `[buffer] [count] X`

114900 Delete the *count* characters before the current character into *buffer*, if specified, and into the
114901 unnamed buffer.

114902 If there are no characters before the current character on the current line, it shall be an error. If
114903 there are less than *count* previous characters on the current line, *count* shall be adjusted to the
114904 number of previous characters on the line.

114905 *Current line:* Unchanged.

114906 *Current column:* Set to (current column – the width of the deleted characters).

114907 **Yank**

114908 *Synopsis:* `[buffer] [count] y motion`

114909 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.

114910 If the motion command is the *y* command repeated:

- 114911 1. The buffer shall be in line mode.
- 114912 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
114913 error.
- 114914 3. The text region shall be from the current line up to and including the next *count* –1 lines.

114915 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

114916 *Current line:* If the motion was from the current cursor position toward the end of the edit buffer,
114917 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region
114918 specified by the motion command.

114919 *Current column:*

- 114920 1. If the motion was from the current cursor position toward the end of the edit buffer,
114921 unchanged.
- 114922 2. Otherwise, if the current line is empty, set to column position 1.
- 114923 3. Otherwise, set to the last column that displays any part of the first character in the file
114924 that is part of the text region specified by the motion command.

114925 **Yank Current Line**114926 *Synopsis:* `[buffer] [count] Y`114927 This command shall be equivalent to the *vi* command:114928 `[buffer] [count] y_`114929 **Redraw Window**114930 If in open mode, the **z** command shall have the Synopsis:114931 *Synopsis:* `[count] z`

114932 If *count* is not specified, it shall default to the **window** edit option `-1`. The **z** command shall be
 114933 equivalent to the *ex z* command, with a type character of `=` and a *count* of *count* `-2`, except that
 114934 the current line and current column shall be set as follows, and the **window** edit option shall not
 114935 be affected. If the calculation for the *count* argument would result in a negative number, the
 114936 *count* argument to the *ex z* command shall be zero. A blank line shall be written after the last line
 114937 is written.

114938 *Current line:* Unchanged.114939 *Current column:* Unchanged.114940 If not in open mode, the **z** command shall have the following Synopsis:114941 *Synopsis:* `[line] z [count] character`

114942 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the
 114943 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

114944 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the
 114945 *ex window* command), and the screen shall be redrawn.

114946 *line* shall be placed as specified by the following characters:114947 `<newline>`, `<carriage-return>`

114948 Place the beginning of the line on the first line of the display.

114949 . Place the beginning of the line in the center of the display. The middle line of the display
 114950 shall be calculated as described for the **M** command.

114951 `-` Place an unspecified portion of the line on the last line of the display.

114952 + If *line* was specified, equivalent to the `<newline>` case. If *line* was not specified, display a
 114953 screen where the first line of the display shall be (current last line) `+1`. If there are no lines
 114954 after the last line in the display, it shall be an error.

114955 ^ If *line* was specified, display a screen where the last line of the display shall contain an
 114956 unspecified portion of the first line of a display that had an unspecified portion of the
 114957 specified line on the last line of the display. If this calculation results in a line before the
 114958 beginning of the edit buffer, display the first screen of the edit buffer.

114959 Otherwise, display a screen where the last line of the display shall contain an unspecified
 114960 portion of (current first line `-1`). If this calculation results in a line before the beginning of
 114961 the edit buffer, it shall be an error.

114962 *Current line*: If *line* and the '^' character were specified:

- 114963 1. If the first screen was displayed as a result of the command attempting to display lines
 114964 before the beginning of the edit buffer: if the first screen was already displayed,
 114965 unchanged; otherwise, set to (current first line -1).
- 114966 2. Otherwise, set to the last line of the display.

114967 If *line* and the '+' character were specified, set to the first line of the display.

114968 Otherwise, if *line* was specified, set to *line*.

114969 Otherwise, unchanged.

114970 *Current column*: Set to non-<blank>.

114971 **Exit**

114972 *Synopsis*: `ZZ`

114973 This command shall be equivalent to the `ex xit` command with no addresses, trailing `!`, or
 114974 filename (see the `ex xit` command).

114975 **Input Mode Commands in vi**

114976 In text input mode, the current line shall consist of zero or more of the following categories, plus
 114977 the terminating <newline>:

- 114978 1. Characters preceding the text input entry point
 114979 Characters in this category shall not be modified during text input mode.
- 114980 2. **autoindent** characters
 114981 **autoindent** characters shall be automatically inserted into each line that is created in text
 114982 input mode, either as a result of entering a <newline> or <carriage-return> while in text
 114983 input mode, or as an effect of the command itself; for example, **O** or **o** (see the `ex`
 114984 **autoindent** command), as if entered by the user.
- 114985 It shall be possible to erase **autoindent** characters with the <control>-D command; it is
 114986 unspecified whether they can be erased by <control>-H, <control>-U, and <control>-W
 114987 characters. Erasing any **autoindent** character turns the glyph into erase-columns and
 114988 deletes the character from the edit buffer, but does not change its representation on the
 114989 screen.
- 114990 3. Text input characters
 114991 Text input characters are the characters entered by the user. Erasing any text input
 114992 character turns the glyph into erase-columns and deletes the character from the edit
 114993 buffer, but does not change its representation on the screen.
- 114994 Each text input character entered by the user (that does not have a special meaning) shall
 114995 be treated as follows:
- 114996 a. The text input character shall be appended to the last character in the edit buffer
 114997 from the first, second, or third categories.
- 114998 b. If there are no erase-columns on the screen, the text input command was the **R**
 114999 command, and characters in the fifth category from the original line follow the
 115000 cursor, the next such character shall be deleted from the edit buffer. If the
 115001 **slowopen** edit option is not set, the corresponding glyph on the screen shall

- 115002 become erase-columns.
- 115003 c. If there are erase-columns on the screen, as many columns as they occupy, or as are
115004 necessary, shall be overwritten to display the text input character. (If only part of a
115005 multi-column glyph is overwritten, the remainder shall be left on the screen, and
115006 continue to be treated as erase-columns; it is unspecified whether the remainder of
115007 the glyph is modified in any way.)
- 115008 d. If additional display line columns are needed to display the text input character:
- 115009 i. If the **slowopen** edit option is set, the text input characters shall be
115010 displayed on subsequent display line columns, overwriting any characters
115011 displayed in those columns.
- 115012 ii. Otherwise, any characters currently displayed on or after the column on the
115013 display line where the text input character is to be displayed shall be
115014 pushed ahead the number of display line columns necessary to display the
115015 rest of the text input character.
- 115016 4. Erase-columns
- 115017 Erase-columns are not logically part of the edit buffer, appearing only on the screen, and
115018 may be overwritten on the screen by subsequent text input characters. When text input
115019 mode ends, all erase-columns shall no longer appear on the screen.
- 115020 Erase-columns are initially the region of text specified by the **c** command (see [Change](#), on
115021 page 3397); however, erasing **autoindent** or text input characters causes the glyphs of the
115022 erased characters to be treated as erase-columns.
- 115023 5. Characters following the text region for the **c** command, or the text input entry point for
115024 all other commands
- 115025 Characters in this category shall not be modified during text input mode, except as
115026 specified in category 3.b. for the **R** text input command, or as <blank> characters deleted
115027 when a <newline> or <carriage-return> is entered.
- 115028 It is unspecified whether it is an error to attempt to erase past the beginning of a line that was
115029 created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an
115030 error, the editor shall behave as if the erasing character was entered immediately after the last
115031 text input character entered on the previous line, and all of the non-<newline> characters on the
115032 current line shall be treated as erase-columns.
- 115033 When text input mode is entered, or after a text input mode character is entered (except as
115034 specified for the special characters below), the cursor shall be positioned as follows:
- 115035 1. On the first column that displays any part of the first erase-column, if one exists
- 115036 2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last
115037 character in the first, second, or third categories, if one exists
- 115038 3. Otherwise, the first column that displays any part of the first character in the fifth
115039 category, if one exists
- 115040 4. Otherwise, the display line column after the last character in the first, second, or third
115041 categories, if one exists
- 115042 5. Otherwise, on column position 1
- 115043 The characters that are updated on the screen during text input mode are unspecified, other than
115044 that the last text input character shall always be updated, and, if the **slowopen** edit option is not

115045 set, the current cursor character shall always be updated.

115046 The following specifications are for command characters entered during text input mode.

115047 NUL

115048 *Synopsis:* NUL

115049 If the first character of the text input is a NUL, the most recently input text shall be input as if
 115050 entered by the user, and then text input mode shall be exited. The text shall be input literally;
 115051 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted
 115052 in any special manner. It is unspecified whether implementations shall support more than 256
 115053 bytes of remembered input text.

115054 <control>-D

115055 *Synopsis:* <control>-D

115056 The <control>-D character shall have no special meaning when in text input mode for a line-
 115057 oriented command (see [Command Descriptions in vi](#), on page 3376).

115058 This command need not be supported on block-mode terminals.

115059 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a '0' or
 115060 '^' character:

- 115061 1. If the cursor is in column position 1, the <control>-D character shall be discarded and no
 115062 further action taken.
- 115063 2. Otherwise, the <control>-D character shall have no special meaning.

115064 If the last input character was a '0', the cursor shall be moved to column position 1.

115065 Otherwise, if the last input character was a '^', the cursor shall be moved to column position 1.
 115066 In addition, the **autoindent** level for the next input line shall be derived from the same line from
 115067 which the **autoindent** level for the current input line was derived.

115068 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the
 115069 *ex* **shiftwidth** command) boundary.

115070 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
 115071 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on
 115072 page 3410).

115073 *Current line:* Unchanged.

115074 *Current column:* Set to 1 if the <control>-D was preceded by a '^' or '0'; otherwise, set to
 115075 (column -1) - ((column -2) % **shiftwidth**).

115076 <control>-H

115077 *Synopsis:* <control>-H

115078 If in text input mode for a line-oriented command, and there are no characters to erase, text
 115079 input mode shall be terminated, no further action shall be done for this command, and the
 115080 current line and column shall be unchanged.

115081 If there are characters other than **autoindent** characters that have been input on the current line
 115082 before the cursor, the cursor shall move back one character.

115083 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is

115084 implementation-defined whether the <control>-H command is an error or if the cursor moves
115085 back one **autoindent** character.

115086 Otherwise, if the cursor is in column position 1 and there are previous lines that have been
115087 input, it is implementation-defined whether the <control>-H command is an error or if it is
115088 equivalent to entering <control>-H after the last input character on the previous input line.

115089 Otherwise, it shall be an error.

115090 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
115091 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on
115092 page 3410).

115093 The current erase character (see *stty*) shall cause an equivalent action to the <control>-H
115094 command, unless the previously inserted character was a <backslash>, in which case it shall be
115095 as if the literal current erase character had been inserted instead of the <backslash>.

115096 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to
115097 line -1.

115098 *Current column*: Set to the first column that displays any portion of the character backed up over.

115099 <newline>

115100 *Synopsis*: <newline>
115101 <carriage-return>
115102 <control>-J
115103 <control>-M

115104 If input was part of a line-oriented command, text input mode shall be terminated and the
115105 command shall continue execution with the input provided.

115106 Otherwise, terminate the current line. If there are no characters other than **autoindent** characters
115107 on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the
115108 **autoindent** characters in the line are modified by entering these characters.

115109 Continue text input mode on a new line appended after the current line. If the **slowopen** edit
115110 option is set, the lines on the screen below the current line shall not be pushed down, but the
115111 first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the
115112 screen below the current line shall be pushed down.

115113 If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be
115114 added as a prefix to the line as described by the *ex* **autoindent** edit option.

115115 All columns after the cursor that are erase-columns (as described in [Input Mode Commands in](#)
115116 [vi](#), on page 3410) shall be discarded.

115117 If the **autoindent** edit option is set, all <blank> characters immediately following the cursor shall
115118 be discarded.

115119 All remaining characters after the cursor shall be transferred to the new line, positioned after
115120 any **autoindent** characters.

115121 *Current line*: Set to current line +1.

115122 *Current column*: Set to the first column that displays any portion of the first character after the
115123 **autoindent** characters on the new line, if any, or the first column position after the last
115124 **autoindent** character, if any, or column position 1.

115125 **<control>-T**115126 *Synopsis:* <control>-T115127 The <control>-T character shall have no special meaning when in text input mode for a line-
115128 oriented command (see [Command Descriptions in vi](#), on page 3376).

115129 This command need not be supported on block-mode terminals.

115130 Behave as if the user entered the minimum number of <blank> characters necessary to move the
115131 cursor forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth**
115132 command) boundary.115133 *Current line:* Unchanged.115134 *Current column:* Set to $column + \mathbf{shiftwidth} - ((column - 1) \% \mathbf{shiftwidth})$.115135 **<control>-U**115136 *Synopsis:* <control>-U115137 If there are characters other than **autoindent** characters that have been input on the current line
115138 before the cursor, the cursor shall move to the first character input after the **autoindent**
115139 characters.115140 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is
115141 implementation-defined whether the <control>-U command is an error or if the cursor moves to
115142 the first column position on the line.115143 Otherwise, if the cursor is in column position 1 and there are previous lines that have been
115144 input, it is implementation-defined whether the <control>-U command is an error or if it is
115145 equivalent to entering <control>-U after the last input character on the previous input line.

115146 Otherwise, it shall be an error.

115147 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
115148 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on
115149 page 3410).115150 The current *kill* character (see *stty*) shall cause an equivalent action to the <control>-U command,
115151 unless the previously inserted character was a <backslash>, in which case it shall be as if the
115152 literal current *kill* character had been inserted instead of the <backslash>.115153 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to
115154 line -1.115155 *Current column:* Set to the first column that displays any portion of the last character backed up
115156 over.115157 **<control>-V**115158 *Synopsis:* <control>-V

115159 <control>-Q

115160 Allow the entry of any subsequent character, other than <control>-J or the <newline>, as a literal
115161 character, removing any special meaning that it may have to the editor in text input mode. If a
115162 <control>-V or <control>-Q is entered before a <control>-J or <newline>, the <control>-V or
115163 <control>-Q character shall be discarded, and the <control>-J or <newline> shall behave as
115164 described in the <newline> command character during input mode.

115165 For purposes of the display only, the editor shall behave as if a '^' character was entered, and

115166 the cursor shall be positioned as if overwriting the '^' character. When a subsequent character
 115167 is entered, the editor shall behave as if that character was entered instead of the original
 115168 <control>-V or <control>-Q character.

115169 *Current line:* Unchanged.

115170 *Current column:* Unchanged.

115171 <control>-W

115172 *Synopsis:* <control>-W

115173 If there are characters other than **autoindent** characters that have been input on the current line
 115174 before the cursor, the cursor shall move back over the last word preceding the cursor (including
 115175 any <blank> characters between the end of the last word and the current cursor); the cursor shall
 115176 not move to before the first character after the end of any **autoindent** characters.

115177 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is
 115178 implementation-defined whether the <control>-W command is an error or if the cursor moves to
 115179 the first column position on the line.

115180 Otherwise, if the cursor is in column position 1 and there are previous lines that have been
 115181 input, it is implementation-defined whether the <control>-W command is an error or if it is
 115182 equivalent to entering <control>-W after the last input character on the previous input line.

115183 Otherwise, it shall be an error.

115184 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
 115185 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on
 115186 page 3410).

115187 *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to
 115188 line -1.

115189 *Current column:* Set to the first column that displays any portion of the last character backed up
 115190 over.

115191 <ESC>

115192 *Synopsis:* <ESC>

115193 If input was part of a line-oriented command:

- 115194 1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to
 115195 command mode. The terminal shall be alerted.
- 115196 2. If <ESC> was entered, text input mode shall be terminated and the command shall
 115197 continue execution with the input provided.

115198 Otherwise, terminate text input mode and return to command mode.

115199 Any **autoindent** characters entered on newly created lines that have no other non-<newline>
 115200 characters shall be deleted.

115201 Any leading **autoindent** and <blank> characters on newly created lines shall be rewritten to be
 115202 the minimum number of <blank> characters possible.

115203 The screen shall be redisplayed as necessary to match the contents of the edit buffer.

115204 *Current line:* Unchanged.

115205 *Current column:*

- 115206 1. If there are text input characters on the current line, the column shall be set to the last
115207 column where any portion of the last text input character is displayed.
- 115208 2. Otherwise, if a character is displayed in the current column, unchanged.
- 115209 3. Otherwise, set to column position 1.

115210 **EXIT STATUS**

115211 The following exit values shall be returned:

- 115212 0 Successful completion.
- 115213 >0 An error occurred.

115214 **CONSEQUENCES OF ERRORS**

115215 When any error is encountered and the standard input is not a terminal device file, *vi* shall not
115216 write the file or return to command or text input mode, and shall terminate with a non-zero exit
115217 status.

115218 Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP
115219 asynchronous event.

115220 Otherwise, when an error is encountered, the editor shall behave as specified in [Command](#)
115221 [Descriptions in vi](#) (on page 3376).

115222 **APPLICATION USAGE**

115223 None.

115224 **EXAMPLES**

115225 None.

115226 **RATIONALE**

115227 See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility
115228 specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been
115229 implemented as a single utility, this is not required by POSIX.1-2017.

115230 It is recognized that portions of *vi* would be difficult, if not impossible, to implement
115231 satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing,
115232 thus it is not a mandatory requirement that such features should work on all terminals. It is the
115233 intention, however, that a *vi* implementation should provide the full set of capabilities on all
115234 terminals capable of supporting them.

115235 Historically, *vi* exited immediately if the standard input was not a terminal. POSIX.1-2017
115236 permits, but does not require, this behavior. An end-of-file condition is not equivalent to an end-
115237 of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.

115238 The text in the STDOUT section reflects the usage of the verb *display* in this section; some
115239 implementations of *vi* use standard output to write to the terminal, but POSIX.1-2017 does not
115240 require that to be the case.

115241 Historically, implementations reverted to open mode if the terminal was incapable of supporting
115242 full visual mode. POSIX.1-2017 requires this behavior. Historically, the open mode of *vi* behaved
115243 roughly equivalently to the visual mode, with the exception that only a single line from the edit
115244 buffer (one “buffer line”) was kept current at any time. This line was normally displayed on the
115245 next-to-last line of a terminal with cursor addressing (and the last line performed its normal
115246 visual functions for line-oriented commands and messages). In addition, some few commands
115247 behaved differently in open mode than in visual mode. POSIX.1-2017 requires conformance to
115248 historical practice.

115249 Historically, *ex* and *vi* implementations have expected text to proceed in the usual
 115250 European/Latin order of left to right, top to bottom. There is no requirement in POSIX.1-2017
 115251 that this be the case. The specification was deliberately written using words like “before”,
 115252 “after”, “first”, and “last” in order to permit implementations to support the natural text order
 115253 of the language.

115254 Historically, lines past the end of the edit buffer were marked with single <tilde> ('~')
 115255 characters; that is, if the one-based display was 20 lines in length, and the last line of the file was
 115256 on line one, then lines 2-20 would contain only a single '~' character.

115257 Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it
 115258 did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the
 115259 bottom of the screen, the screen lines where the line would have been displayed were displayed
 115260 as single '@' characters, instead of displaying part of the line. POSIX.1-2017 permits, but does
 115261 not require, this behavior. Implementations are encouraged to attempt always to display a
 115262 complete line at the bottom of the screen when doing scrolling or screen positioning by buffer
 115263 lines.

115264 Historically, lines marked with '@' were also used to minimize output to dumb terminals over
 115265 slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen
 115266 that were not close to the cursor were simply marked with an '@' sign instead of being updated
 115267 to match the current text. POSIX.1-2017 permits, but does not require this feature because it is
 115268 used ever less frequently as terminals become smarter and connections are faster.

115269 Initialization in *ex* and *vi*

115270 Historically, *vi* always had a line in the edit buffer, even if the edit buffer was “empty”. For
 115271 example:

- 115272 1. The *ex* command = executed from visual mode wrote “1” when the buffer was empty.
- 115273 2. Writes from visual mode of an empty edit buffer wrote files of a single character (a
 115274 <newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.
- 115275 3. Put and read commands into an empty edit buffer left an empty line at the top of the edit
 115276 buffer.

115277 For consistency, POSIX.1-2017 does not permit any of these behaviors.

115278 Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was
 115279 modified if it was not originally set. POSIX.1-2017 does not permit this behavior.

115280 Command Descriptions in *vi*

115281 Motion commands are among the most complicated aspects of *vi* to describe. With some
 115282 exceptions, the text region and buffer type effect of a motion command on a *vi* command are
 115283 described on a case-by-case basis. The descriptions of text regions in POSIX.1-2017 are not
 115284 intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to a
 115285 region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks can
 115286 be in either direction, and, if the **wrapsan** option is set, so can movements to search points.
 115287 Historically, lines are always stored into buffers in text order; that is, from the start of the edit
 115288 buffer to the end. POSIX.1-2017 requires conformance to historical practice.

115289 Historically, command counts were applied to any associated motion, and were multiplicative to
 115290 any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as **c6w**.
 115291 POSIX.1-2017 requires this behavior. Historically, *vi* commands that used bigwords, words,
 115292 paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only

115293 <blank> characters, inconsistently. Some commands treated them as a single entity, while others
 115294 treated each line separately. For example, the **w**, **W**, and **B** commands treated groups of empty
 115295 lines as individual words; that is, the command would move the cursor to each new empty line.
 115296 The **e** and **E** commands treated groups of empty lines as a single word; that is, the first use
 115297 would move past the group of lines. The **b** command would just beep at the user, or if done from
 115298 the start of the line as a motion command, fail in unexpected ways. If the lines contained only (or
 115299 ended with) <blank> characters, the **w** and **W** commands would just beep at the user, the **E** and
 115300 **e** commands would treat the group as a single word, and the **B** and **b** commands would treat the
 115301 lines as individual words. For consistency and simplicity of specification, POSIX.1-2017 requires
 115302 that all *vi* commands treat groups of empty or blank lines as a single entity, and that movement
 115303 through lines ending with <blank> characters be consistent with other movements.

115304 Historically, *vi* documentation indicated that any number of double-quotes were skipped after
 115305 punctuation marks at sentence boundaries; however, implementations only skipped single-
 115306 quotes. POSIX.1-2017 requires both to be skipped.

115307 Historically, the first and last characters in the edit buffer were word boundaries. This historical
 115308 practice is required by POSIX.1-2017.

115309 Historically, *vi* attempted to update the minimum number of columns on the screen possible,
 115310 which could lead to misleading information being displayed. POSIX.1-2017 makes no
 115311 requirements other than that the current character being entered is displayed correctly, leaving
 115312 all other decisions in this area up to the implementation.

115313 Historically, lines were arbitrarily folded between columns of any characters that required
 115314 multiple column positions on the screen, with the exception of tabs, which terminated at the
 115315 right-hand margin. POSIX.1-2017 permits the former and requires the latter. Implementations
 115316 that do not arbitrarily break lines between columns of characters that occupy multiple column
 115317 positions should not permit the cursor to rest on a column that does not contain any part of a
 115318 character.

115319 The historical *vi* had a problem in that all movements were by buffer lines, not by display or
 115320 screen lines. This is often the right thing to do; for example, single line movements, such as **j** or
 115321 **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only
 115322 make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling
 115323 commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using buffer lines
 115324 in these cases can result in completely random motion; for example, **1<control>-D** can result in a
 115325 completely changed screen, without any overlap. This is clearly not what the user wanted. The
 115326 problem is even worse in the case of the **H**, **L**, and **M** commands — as they position the cursor at
 115327 the first non-<blank> of the line, they may all refer to the same location in large lines, and will
 115328 result in no movement at all.

115329 In addition, if the line is larger than the screen, using buffer lines can make it impossible to
 115330 display parts of the line — there are not any commands that do not display the beginning of the
 115331 line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at the
 115332 same time, the user suffers. Finally, the page and half-page scrolling commands historically
 115333 moved to the first non-<blank> in the new line. If the line is approximately the same size as the
 115334 screen, this is inadequate because the cursor before and after a <control>-D command will refer
 115335 to the same location on the screen.

115336 Implementations of *ex* and *vi* exist that do not have these problems because the relevant
 115337 commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**,
 115338 and **M**) operate on display (screen) lines, not (edit) buffer lines.

115339 POSIX.1-2017 does not permit this behavior by default because the standard developers believed
 115340 that users would find it too confusing. However, historical practice has been relaxed. For

115341 example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part of a
115342 line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of the
115343 line was displayed, and the screen lines corresponding to the line contained single '@'
115344 characters. This behavior is permitted, but not required by POSIX.1-2017, so that it is possible for
115345 implementations to support long lines in small screens more reasonably without changing the
115346 commands to be oriented to the display (instead of oriented to the buffer). POSIX.1-2017 also
115347 permits implementations to refuse to edit any edit buffer containing a line that will not fit on the
115348 screen in its entirety.

115349 The display area (for example, the value of the **window** edit option) has historically been
115350 “grown”, or expanded, to display new text when local movements are done in displays where
115351 the number of lines displayed is less than the maximum possible. Expansion has historically
115352 been the first choice, when the target line is less than the maximum possible expansion value
115353 away. Scrolling has historically been the next choice, done when the target line is less than half a
115354 display away, and otherwise, the screen was redrawn. There were exceptions, however, in that *ex*
115355 commands generally always caused the screen to be redrawn. POSIX.1-2017 does not specify a
115356 standard behavior because there may be external issues, such as connection speed, the number
115357 of characters necessary to redraw as opposed to scroll, or terminal capabilities that
115358 implementations will have to accommodate.

115359 The current line in POSIX.1-2017 maps one-to-one to a buffer line in the file. The current column
115360 does not. There are two different column values that are described by POSIX.1-2017. The first is
115361 the current column value as set by many of the *vi* commands. This value is remembered for the
115362 lifetime of the editor. The second column value is the actual position on the screen where the
115363 cursor rests. The two are not always the same. For example, when the cursor is backed by a
115364 multi-column character, the actual cursor position on the screen has historically been the last
115365 column of the character in command mode, and the first column of the character in input mode.

115366 Commands that set the current line, but that do not set the current cursor value (for example, **j**
115367 and **k**) attempt to get as close as possible to the remembered column position, so that the cursor
115368 tends to restrict itself to a vertical column as the user moves around in the edit buffer.
115369 POSIX.1-2017 requires conformance to historical practice, requiring that the display location of
115370 the cursor on the display line be adjusted from the current column value as necessary to support
115371 this historical behavior.

115372 Historically, only a single line (and for some terminals, a single line minus 1 column) of
115373 characters could be entered by the user for the line-oriented commands; that is, **;**, **!**, **/**, or **?**.
115374 POSIX.1-2017 permits, but does not require, this limitation.

115375 Historically, “soft” errors in *vi* caused the terminal to be alerted, but no error message was
115376 displayed. As a general rule, no error message was displayed for errors in command execution
115377 in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when
115378 a searched-for object was not found. Examples of soft errors included **h** at the left margin,
115379 <control>-**B** or **[** at the beginning of the file, **2G** at the end of the file, and so on. In addition,
115380 errors such as **%**, **]**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well.
115381 Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n**
115382 displayed an error message if no previous regular expression had been specified, and **;** did not
115383 display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in
115384 this area might reasonably be based on a runtime evaluation of the speed of a network
115385 connection. Finally, some implementations have provided error messages for soft errors in order
115386 to assist naive users, based on the value of a verbose edit option. POSIX.1-2017 does not list
115387 specific errors for which an error message shall be displayed. Implementations should conform
115388 to historical practice in the absence of any strong reason to diverge.

115389 Page Backwards

115390 The <control>-B and <control>-F commands historically considered it an error to attempt to
 115391 page past the beginning or end of the file, whereas the <control>-D and <control>-U commands
 115392 simply moved to the beginning or end of the file. For consistency, POSIX.1-2017 requires the
 115393 latter behavior for all four commands. All four commands still consider it an error if the current
 115394 line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of the file.
 115395 Historically, the <control>-B and <control>-F commands skip two lines in order to include
 115396 overlapping lines when a single command is entered. This makes less sense in the presence of a
 115397 *count*, as there will be, by definition, no overlapping lines. The actual calculation used by
 115398 historical implementations of the *vi* editor for <control>-B was:

```
115399 ((current first line) - count x (window edit option)) +2
```

115400 and for <control>-F was:

```
115401 ((current first line) + count x (window edit option)) -2
```

115402 This calculation does not work well when intermixing commands with and without counts; for
 115403 example, **3<control>-F is not equivalent to entering the <control>-F command three times,**
 115404 **and is not reversible by entering the <control>-B command three times. For consistency with**
 115405 **other *vi* commands that take counts, POSIX.1-2017 requires a different calculation.**

115406 Scroll Forward

115407 The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll**
 115408 command. 4BSD used:

```
115409 ((window edit option) +1) /2
```

115410 while System V used the value of the **scroll** edit option. The System V version is specified by
 115411 POSIX.1-2017 because the standard developers believed that it was more intuitive and permitted
 115412 the user a method of setting the scroll value initially without also setting the number of lines that
 115413 are displayed.

115414 Scroll Forward by Line

115415 Historically, the <control>-E and <control>-Y commands considered it an error if the last and
 115416 first lines, respectively, were already on the screen. POSIX.1-2017 requires conformance to
 115417 historical practice. Historically, the <control>-E and <control>-Y commands had no effect in
 115418 open mode. For simplicity and consistency of specification, POSIX.1-2017 requires that they
 115419 behave as usual, albeit with a single line screen.

115420 Clear and Redisplay

115421 The historical <control>-L command refreshed the screen exactly as it was supposed to be
 115422 currently displayed, replacing any '@' characters for lines that had been deleted but not
 115423 updated on the screen with refreshed '@' characters. The intent of the <control>-L command is
 115424 to refresh when the screen has been accidentally overwritten; for example, by a **write** command
 115425 from another user, or modem noise.

115426

Redraw Screen

115427

115428

115429

115430

115431

115432

The historical <control>-R command redisplayed only when necessary to update lines that had been deleted but not updated on the screen and that were flagged with '@' characters. There is no requirement that the screen be in any way refreshed if no lines of this form are currently displayed. POSIX.1-2017 permits implementations to extend this command to refresh lines on the screen flagged with '@' characters because they are too long to be displayed in the current framework; however, the current line and column need not be modified.

115433

Search for tagstring

115434

115435

115436

115437

115438

Historically, the first non-<blank> at or after the cursor was the first character, and all subsequent characters that were word characters, up to the end of the line, were included. For example, with the cursor on the leading <space> or on the '#' character in the text "#bar@", the tag was "#bar". On the character 'b' it was "bar", and on the 'a' it was "ar". POSIX.1-2017 requires this behavior.

115439

Replace Text with Results from Shell Command

115440

115441

115442

115443

115444

Historically, the <, >, and ! commands considered most cursor motions other than line-oriented motions an error; for example, the command >/foo<CR> succeeded, while the command >I failed, even though the text region described by the two commands might be identical. For consistency, all three commands only consider entire lines and not partial lines, and the region is defined as any line that contains a character that was specified by the motion.

115445

Move to Matching Character

115446

115447

115448

Other matching characters have been left implementation-defined in order to allow extensions such as matching '<' and '>' for searching HTML, or #ifdef, #else, and #endif for searching C source.

115449

Repeat Substitution

115450

115451

POSIX.1-2017 requires that any c and g flags specified to the previous substitute command be ignored; however, the r flag may still apply, if supported by the implementation.

115452

Return to Previous (Context or Section)

115453

115454

115455

115456

115457

115458

115459

115460

115461

115462

115463

The [,], (,), {, and } commands are all affected by "section boundaries", but in some historical implementations not all of the commands recognize the same section boundaries. This is a bug, not a feature, and a unique section-boundary algorithm was not described for each command. One special case that is preserved is that the sentence command moves to the end of the last line of the edit buffer while the other commands go to the beginning, in order to preserve the traditional character cut semantics of the sentence command. Historically, vi section boundaries at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit buffer if one exists. To increase consistency with other section locations, this has been simplified by POSIX.1-2017 to the first character of the first and last lines of the edit buffer, or the first and the last lines of the edit buffer if they are empty.

115464

115465

115466

115467

115468

Sentence boundaries were problematic in the historical vi. They were not only the boundaries as defined for the section and paragraph commands, but they were the first non-<blank> that occurred after those boundaries, as well. Historically, the vi section commands were documented as taking an optional window size as a count preceding the command. This was not implemented in historical versions, so POSIX.1-2017 requires that the count repeat the command,

115469 for consistency with other *vi* commands.

115470 **Repeat**

115471 Historically, mapped commands other than text input commands could not be repeated using
115472 the **period** command. POSIX.1-2017 requires conformance to historical practice.

115473 The restrictions on the interpretation of special characters (for example, <control>-H) in the
115474 repetition of text input mode commands is intended to match historical practice. For example,
115475 given the input sequence:

```
115476 iab<control>-H<control>-H<control>-Hdef<escape>
```

115477 the user should be informed of an error when the sequence is first entered, but not during a
115478 command repetition. The character <control>-T is specifically exempted from this restriction.
115479 Historical implementations of *vi* ignored <control>-T characters that were input in the original
115480 command during command repetition. POSIX.1-2017 prohibits this behavior.

115481 **Find Regular Expression**

115482 Historically, commands did not affect the line searched to or from if the motion command was a
115483 search (*/*, *?*, **N**, **n**) and the final position was the start/end of the line. There were some special
115484 cases and *vi* was not consistent. POSIX.1-2017 does not permit this behavior, for consistency.
115485 Historical implementations permitted but were unable to handle searches as motion commands
115486 that wrapped (that is, due to the edit option **wrapsan**) to the original location. POSIX.1-2017
115487 requires that this behavior be treated as an error.

115488 Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode.
115489 POSIX.1-2017 requires conformance to historical practice.

115490 Historically, in open mode, a **z** specified to a search command redisplayed the current line
115491 instead of displaying the current screen with the current line highlighted. For consistency and
115492 simplicity of specification, POSIX.1-2017 does not permit this behavior.

115493 Historically, trailing **z** commands were permitted and ignored if entered as part of a search used
115494 as a motion command. For consistency and simplicity of specification, POSIX.1-2017 does not
115495 permit this behavior.

115496 **Execute an ex Command**

115497 Historically, *vi* implementations restricted the commands that could be entered on the colon
115498 command line (for example, **append** and **change**), and some other commands were known to
115499 cause them to fail catastrophically. For consistency, POSIX.1-2017 does not permit these
115500 restrictions. When executing an *ex* command by entering `:`, it is not possible to enter a <newline>
115501 as part of the command because it is considered the end of the command. A different approach
115502 is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with
115503 the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist. So, for
115504 example, the following is valid:

```
115505 Q
115506 s/break here/break\
115507 here/
115508 vi
```

115509 POSIX.1-2017 requires that, if the *ex* command overwrites any part of the screen that would be
115510 erased by a refresh, *vi* pauses for a character from the user. Historically, this character could be
115511 any character; for example, a character input by the user before the message appeared, or even a

115512 mapped character. This is probably a bug, but implementations that have tried to be more
115513 rigorous by requiring that the user enter a specific character, or that the user enter a character
115514 after the message was displayed, have been forced by user indignation back into historical
115515 behavior. POSIX.1-2017 requires conformance to historical practice.

115516 **Shift Left (Right)**

115517 Refer to the Rationale for the ! and / commands. Historically, the < and > commands sometimes
115518 moved the cursor to the first non-<blank> (for example if the command was repeated or with _
115519 as the motion command), and sometimes left it unchanged. POSIX.1-2017 does not permit this
115520 inconsistency, requiring instead that the cursor always move to the first non-<blank>.
115521 Historically, the < and > commands did not support buffer arguments, although some
115522 implementations allow the specification of an optional buffer. This behavior is neither required
115523 nor disallowed by POSIX.1-2017.

115524 **Execute**

115525 Historically, buffers could execute other buffers, and loops, infinite and otherwise, were
115526 possible. POSIX.1-2017 requires conformance to historical practice. The **buffer* syntax of *ex* is not
115527 required in *vi*, because it is not historical practice and has been used in some *vi* implementations
115528 to support additional scripting languages.

115529 **Reverse Case**

115530 Historically, the ~ command ignored any associated *count*, and acted only on the characters in the
115531 current line. For consistency with other *vi* commands, POSIX.1-2017 requires that an associated
115532 *count* act on the next *count* characters, and that the command move to subsequent lines if
115533 warranted by *count*, to make it possible to modify large pieces of text in a reasonably efficient
115534 manner. There exist *vi* implementations that optionally require an associated motion command
115535 for the ~ command. Implementations supporting this functionality are encouraged to base it on
115536 the **tildedop** edit option and handle the text regions and cursor positioning identically to the
115537 **yank** command.

115538 **Append**

115539 Historically, *counts* specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line
115540 *count* times, and did not repeat the subsequent lines of the input text. POSIX.1-2017 requires that
115541 the entire text input be repeated *count* times.

115542 **Move Backward to Preceding Word**

115543 Historically, *vi* became confused if word commands were used as motion commands in empty
115544 files. POSIX.1-2017 requires that this be an error. Historical implementations of *vi* had a large
115545 number of bugs in the word movement commands, and they varied greatly in behavior in the
115546 presence of empty lines, ``words'' made up of a single character, and lines containing only
115547 <blank> characters. For consistency and simplicity of specification, POSIX.1-2017 does not
115548 permit this behavior.

115549 Change to End-of-Line

115550 Some historical implementations of the **C** command did not behave as described by
115551 POSIX.1-2017 when the **\$** key was remapped because they were implemented by pushing the **\$**
115552 key onto the input queue and reprocessing it. POSIX.1-2017 does not permit this behavior.
115553 Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric buffers. For
115554 consistency and simplicity of specification, POSIX.1-2017 requires that they behave like their
115555 respective **c** commands in all respects.

115556 Delete

115557 Historically, lines in open mode that were deleted were scrolled up, and an **@** glyph written over
115558 the beginning of the line. In the case of terminals that are incapable of the necessary cursor
115559 motions, the editor erased the deleted line from the screen. POSIX.1-2017 requires conformance
115560 to historical practice; that is, if the terminal cannot display the **'@'** character, the line cannot
115561 remain on the screen.

115562 Delete to End-of-Line

115563 Some historical implementations of the **D** command did not behave as described by
115564 POSIX.1-2017 when the **\$** key was remapped because they were implemented by pushing the **\$**
115565 key onto the input queue and reprocessing it. POSIX.1-2017 does not permit this behavior.

115566 Join

115567 An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent. POSIX.1-2017
115568 requires conformance to historical practice. The *vi* **J** command is specified in terms of the *ex* **join**
115569 command with an *ex* command *count* value. The address correction for a *count* that is past the
115570 end of the edit buffer is necessary for historical compatibility for both *ex* and *vi*.

115571 Mark Position

115572 Historical practice is that only lowercase letters, plus backquote and single-quote, could be used
115573 to mark a cursor position. POSIX.1-2017 requires conformance to historical practice, but
115574 encourages implementations to support other characters as marks as well.

115575 Repeat Regular Expression Find (Forward and Reverse)

115576 Historically, the **N** and **n** commands could not be used as motion components for the **c**
115577 command. With the exception of the **cN** command, which worked if the search crossed a line
115578 boundary, the text region would be discarded, and the user would not be in text input mode. For
115579 consistency and simplicity of specification, POSIX.1-2017 does not permit this behavior.

115580 Insert Empty Line (Below and Above)

115581 Historically, counts to the **O** and **o** commands were used as the number of physical lines to
115582 open, if the terminal was dumb and the **slowopen** option was not set. This was intended to
115583 minimize traffic over slow connections and repainting for dumb terminals. POSIX.1-2017 does
115584 not permit this behavior, requiring that a *count* to the open command behave as for other text
115585 input commands. This change to historical practice was made for consistency, and because a
115586 superset of the functionality is provided by the **slowopen** edit option.

115587 Put from Buffer (Following and Before)

115588 Historically, *counts* to the **p** and **P** commands were ignored if the buffer was a line mode buffer,
115589 but were (mostly) implemented as described in POSIX.1-2017 if the buffer was a character mode
115590 buffer. Because implementations exist that do not have this limitation, and because pasting lines
115591 multiple times is generally useful, POSIX.1-2017 requires that *count* be supported for all **p** and **P**
115592 commands.

115593 Historical implementations of *vi* were widely known to have major problems in the **p** and **P**
115594 commands, particularly when unusual regions of text were copied into the edit buffer. The
115595 standard developers viewed these as bugs, and they are not permitted for consistency and
115596 simplicity of specification.

115597 Historically, a **P** or **p** command (or an *ex* **put** command executed from open or visual mode)
115598 executed in an empty file, left an empty line as the first line of the file. For consistency and
115599 simplicity of specification, POSIX.1-2017 does not permit this behavior.

115600 Replace Character

115601 Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as
115602 arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return>
115603 argument, for which it replaced *count* characters with a single <newline>. POSIX.1-2017 does
115604 not permit these inconsistencies.

115605 Historically, the **r** command permitted the <control>-V escaping of entered characters, such as
115606 <ESC> and the <carriage-return>; however, it required two leading <control>-V characters
115607 instead of one. POSIX.1-2017 requires that this be changed for consistency with the other text
115608 input commands of *vi*.

115609 Historically, it is an error to enter the **r** command if there are less than *count* characters at or after
115610 the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r**
115611 command on empty lines, it would require that too large a *count* be adjusted to match the
115612 number of characters at or after the cursor for consistency, which is sufficiently different from
115613 historical practice to be avoided. POSIX.1-2017 requires conformance to historical practice.

115614 Replace Characters

115615 Historically, if there were **autoindent** characters in the line on which the **R** command was run,
115616 and **autoindent** was set, the first <newline> would be properly indented and no characters
115617 would be replaced by the <newline>. Each additional <newline> would replace *n* characters,
115618 where *n* was the number of characters that were needed to indent the rest of the line to the
115619 proper indentation level. This behavior is a bug and is not permitted by POSIX.1-2017.

115620 Undo

115621 Historical practice for cursor positioning after undoing commands was mixed. In most cases,
115622 when undoing commands that affected a single line, the cursor was moved to the start of added
115623 or changed text, or immediately after deleted text. However, if the user had moved from the line
115624 being changed, the column was either set to the first non-<blank>, returned to the origin of the
115625 command, or remained unchanged. When undoing commands that affected multiple lines or
115626 entire lines, the cursor was moved to the first character in the first line restored. As an example
115627 of how inconsistent this was, a search, followed by an **o** text input command, followed by an
115628 **undo** would return the cursor to the location where the **o** command was entered, but a **cw**
115629 command followed by an **o** command followed by an **undo** would return the cursor to the first
115630 non-<blank> of the line. POSIX.1-2017 requires the most useful of these behaviors, and discards
115631 the least useful, in the interest of consistency and simplicity of specification.

115632 **Yank**

115633 Historically, the **yank** command did not move to the end of the motion if the motion was in the
 115634 forward direction. It moved to the end of the motion if the motion was in the backward
 115635 direction, except for the **_** command, or for the **G** and **'** commands when the end of the motion
 115636 was on the current line. This was further complicated by the fact that for a number of motion
 115637 commands, the **yank** command moved the cursor but did not update the screen; for example, a
 115638 subsequent command would move the cursor from the end of the motion, even though the
 115639 cursor on the screen had not reflected the cursor movement for the **yank** command.
 115640 POSIX.1-2017 requires that all **yank** commands associated with backward motions move the
 115641 cursor to the end of the motion for consistency, and specifically, to make **'** commands as motions
 115642 consistent with search patterns as motions.

115643 **Yank Current Line**

115644 Some historical implementations of the **Y** command did not behave as described by
 115645 POSIX.1-2017 when the **'_'** key was remapped because they were implemented by pushing the
 115646 **'_'** key onto the input queue and reprocessing it. POSIX.1-2017 does not permit this behavior.

115647 **Redraw Window**

115648 Historically, the **z** command always redrew the screen. This is permitted but not required by
 115649 POSIX.1-2017, because of the frequent use of the **z** command in macros such as **map n nz.** for
 115650 screen positioning, instead of its use to change the screen size. The standard developers
 115651 believed that expanding or scrolling the screen offered a better interface for users. The ability to
 115652 redraw the screen is preserved if the optional new window size is specified, and in the
 115653 **<control>-L** and **<control>-R** commands.

115654 The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen
 115655 that ended with the specified line is displayed. POSIX.1-2017 requires conformance to historical
 115656 practice.

115657 Historically, the **z** command would not display a partial line at the top or bottom of the screen. If
 115658 the partial line would normally have been displayed at the bottom of the screen, the command
 115659 worked, but the partial line was replaced with **'@'** characters. If the partial line would normally
 115660 have been displayed at the top of the screen, the command would fail. For consistency and
 115661 simplicity of specification, POSIX.1-2017 does not permit this behavior.

115662 Historically, the **z** command with a line specification of 1 ignored the command. For consistency
 115663 and simplicity of specification, POSIX.1-2017 does not permit this behavior.

115664 Historically, the **z** command did not set the cursor column to the first non-**<blank>** for the
 115665 character if the first screen was to be displayed, and was already displayed. For consistency and
 115666 simplicity of specification, POSIX.1-2017 does not permit this behavior.

115667 **Input Mode Commands in vi**

115668 Historical implementations of **vi** did not permit the user to erase more than a single line of input,
 115669 or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent**
 115670 characters. As there exist implementations of **vi** that do not have these limitations, both
 115671 behaviors are permitted, but only historical practice is required. In the case of these extensions,
 115672 **vi** is required to pause at the **autoindent** and previous line boundaries.

115673 Historical implementations of **vi** updated only the portion of the screen where the current cursor
 115674 character was displayed. For example, consider the **vi** input keystrokes:

115675 `iabcd<escape>0C<tab>`

115676 Historically, the <tab> would overwrite the characters "abcd" when it was displayed. Other
 115677 implementations replace only the 'a' character with the <tab>, and then push the rest of the
 115678 characters ahead of the cursor. Both implementations have problems. The historical
 115679 implementation is probably visually nicer for the above example; however, for the keystrokes:

115680 iabcd<ESC>OR<tab><ESC>

115681 the historical implementation results in the string "bcd" disappearing and then magically
 115682 reappearing when the <ESC> character is entered. POSIX.1-2017 requires the former behavior
 115683 when overwriting erase-columns ‡that is, overwriting characters that are no longer logically
 115684 part of the edit buffer—and the latter behavior otherwise.

115685 Historical implementations of *vi* discarded the <control>-D and <control>-T characters when
 115686 they were entered at places where their command functionality was not appropriate.
 115687 POSIX.1-2017 requires that the <control>-T functionality always be available, and that
 115688 <control>-D be treated as any other key when not operating on **autoindent** characters.

115689 NUL

115690 Some historical implementations of *vi* limited the number of characters entered using the NUL
 115691 input character to 256 bytes. POSIX.1-2017 permits this limitation; however, implementations are
 115692 encouraged to remove this limit.

115693 <control>-D

115694 See also Rationale for the input mode command <newline>. The hidden assumptions in the
 115695 <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>
 115696 characters take up a single column on the screen and that <tab> characters are comprised of an
 115697 integral number of <space> characters.

115698 <newline>

115699 Implementations are permitted to rewrite **autoindent** characters in the line when <newline>,
 115700 <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are
 115701 used, because historical implementations have both done so and found it necessary to do so. For
 115702 example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and
 115703 **shiftwidth** set to 3, will result in the <tab> being replaced by several <space> characters.

115704 <control>-T

115705 See also the Rationale for the input mode command <newline>. Historically, <control>-T only
 115706 worked if no non-<blank> characters had yet been input in the current input line. In addition,
 115707 the characters inserted by <control>-T were treated as **autoindent** characters, and could not be
 115708 erased using normal user erase characters. Because implementations exist that do not have
 115709 these limitations, and as moving to a column boundary is generally useful, POSIX.1-2017
 115710 requires that both limitations be removed.

- 115711 **<control>-V**
- 115712 Historically, *vi* used **^V**, regardless of the value of the literal-next character of the terminal.
115713 POSIX.1-2017 requires conformance to historical practice.
- 115714 The uses described for <control>-V can also be accomplished with <control>-Q, which is useful
115715 on terminals that use <control>-V for the down-arrow function. However, most historical
115716 implementations use <control>-Q for the *termios* START character, so the editor will generally
115717 not receive the <control>-Q unless **stty ixon** mode is set to off. (In addition, some historical
115718 implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to
115719 off.) Any of the command characters described in POSIX.1-2017 can be made ineffective by their
115720 selection as *termios* control characters, using the *stty* utility or other methods described in the
115721 System Interfaces volume of POSIX.1-2017.
- 115722 **<ESC>**
- 115723 Historically, SIGINT alerted the terminal when used to end input mode. This behavior is
115724 permitted, but not required, by POSIX.1-2017.
- 115725 **FUTURE DIRECTIONS**
- 115726 None.
- 115727 **SEE ALSO**
- 115728 *ed, ex, stty*
- 115729 XBD [Section 12.2](#) (on page 216)
- 115730 **CHANGE HISTORY**
- 115731 First released in Issue 2.
- 115732 **Issue 5**
- 115733 The FUTURE DIRECTIONS section is added.
- 115734 **Issue 6**
- 115735 This utility is marked as part of the User Portability Utilities option.
- 115736 The APPLICATION USAGE section is added.
- 115737 The obsolescent SYNOPSIS is removed.
- 115738 The following new requirements on POSIX implementations derive from alignment with the
115739 Single UNIX Specification:
- 115740 The **reindent** command description is added.
- 115741 The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft
115742 standard.
- 115743 IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.
- 115744 IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in
115745 a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**.
- 115746 The **-l** option is removed.
- 115747 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/41 is applied, adding *[count]* to the
115748 Synopsis for **[[**.
- 115749 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/42 is applied, adding *[count]* to the
115750 Synopsis for **]].**

115751 **Issue 7**

- 115752 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized
115753 as an option delimiter in the OPTIONS section.
- 115754 Austin Group Interpretation 1003.1-2001 #087 is applied, updating the Put from Buffer Before (P)
115755 command description to address multi-line requirements.
- 115756 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 115757 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0202 [812] is applied.

115758 **NAME**

115759 wait — await process completion

115760 **SYNOPSIS**115761 wait [*pid*...]115762 **DESCRIPTION**

115763 When an asynchronous list (see [Section 2.9.3.1](#), on page 2370) is started by the shell, the process
 115764 ID of the last command in each element of the asynchronous list shall become known in the
 115765 current shell execution environment; see [Section 2.12](#) (on page 2381).

115766 If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the
 115767 invoking shell have terminated and exit with a zero exit status.

115768 If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall
 115769 wait until all of them have terminated. If one or more *pid* operands are specified that represent
 115770 unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with
 115771 exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process
 115772 requested by the last *pid* operand.

115773 The known process IDs are applicable only for invocations of *wait* in the current shell execution
 115774 environment.

115775 **OPTIONS**

115776 None.

115777 **OPERANDS**

115778 The following operand shall be supported:

115779 *pid* One of the following:

- 115780 1. The unsigned decimal integer process ID of a command, for which the
115781 utility is to wait for the termination.
- 115782 2. A job control job ID (see [XBD Section 3.204](#), on page 66) that identifies a
115783 background process group to be waited for. The job control job ID notation
115784 is applicable only for invocations of *wait* in the current shell execution
115785 environment; see [Section 2.12](#) (on page 2381). The exit status of *wait* shall be
115786 determined by the last command in the pipeline.

115787 **Note:** The job control job ID type of *pid* is only available on systems supporting
 115788 the User Portability Utilities option.

115789 **STDIN**

115790 Not used.

115791 **INPUT FILES**

115792 None.

115793 **ENVIRONMENT VARIABLES**115794 The following environment variables shall affect the execution of *wait*:

115795 *LANG* Provide a default value for the internationalization variables that are unset or null.
 115796 (See [XBD Section 8.2](#) (on page 174) for the precedence of internationalization
 115797 variables used to determine the values of locale categories.)

115798 *LC_ALL* If set to a non-empty string value, override the values of all the other
 115799 internationalization variables.

- 115800 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 115801 characters (for example, single-byte as opposed to multi-byte characters in
 115802 arguments).
- 115803 **LC_MESSAGES**
 115804 Determine the locale that should be used to affect the format and contents of
 115805 diagnostic messages written to standard error.
- 115806 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 115807 **ASYNCHRONOUS EVENTS**
 115808 Default.
- 115809 **STDOUT**
 115810 Not used.
- 115811 **STDERR**
 115812 The standard error shall be used only for diagnostic messages.
- 115813 **OUTPUT FILES**
 115814 None.
- 115815 **EXTENDED DESCRIPTION**
 115816 None.
- 115817 **EXIT STATUS**
 115818 If one or more operands were specified, all of them have terminated or were not known by the
 115819 invoking shell, and the status of the last operand specified is known, then the exit status of *wait*
 115820 shall be the exit status information of the command indicated by the last operand specified. If
 115821 the process terminated abnormally due to the receipt of a signal, the exit status shall be greater
 115822 than 128 and shall be distinct from the exit status generated by other signals, but the exact value
 115823 is unspecified. (See the *kill* **-I** option.) Otherwise, the *wait* utility shall exit with one of the
 115824 following values:
- 115825 0 The *wait* utility was invoked with no operands and all process IDs known by the
 115826 invoking shell have terminated.
- 115827 1-126 The *wait* utility detected an error.
- 115828 127 The command identified by the last *pid* operand specified is unknown.
- 115829 **CONSEQUENCES OF ERRORS**
 115830 Default.
- 115831 **APPLICATION USAGE**
 115832 On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility
 115833 execution environment, such as one of the following:
- 115834 (wait)
 115835 nohup wait ...
 115836 find . -exec wait ... \;
- 115837 it returns immediately because there are no known process IDs to wait for in those
 115838 environments.
- 115839 Historical implementations of interactive shells have discarded the exit status of terminated
 115840 background processes before each shell prompt. Therefore, the status of background processes
 115841 was usually lost unless it terminated while *wait* was waiting for it. This could be a serious
 115842 problem when a job that was expected to run for a long time actually terminated quickly with a
 115843 syntax or initialization error because the exit status returned was usually zero if the requested

115844 process ID was not found. This volume of POSIX.1-2017 requires the implementation to keep the
 115845 status of terminated jobs available until the status is requested, so that scripts like:

```
115846 j1&
115847 p1=$!
115848 j2&
115849 wait $p1
115850 echo Job 1 exited with status $?
115851 wait $!
115852 echo Job 2 exited with status $?
```

115853 work without losing status on any of the jobs. The shell is allowed to discard the status of any
 115854 process if it determines that the application cannot get the process ID for that process from the
 115855 shell. It is also required to remember only {CHILD_MAX} number of processes in this way. Since
 115856 the only way to get the process ID from the shell is by using the '!' shell parameter, the shell is
 115857 allowed to discard the status of an asynchronous list if "\$!" was not referenced before another
 115858 asynchronous list was started. (This means that the shell only has to keep the status of the last
 115859 asynchronous list started if the application did not reference "\$!". If the implementation of the
 115860 shell is smart enough to determine that a reference to "\$!" was not saved anywhere that the
 115861 application can retrieve it later, it can use this information to trim the list of saved information.
 115862 Note also that a successful call to *wait* with no operands discards the exit status of all
 115863 asynchronous lists.)

115864 If the exit status of *wait* is greater than 128, there is no way for the application to know if the
 115865 waited-for process exited with that value or was killed by a signal. Since most utilities exit with
 115866 small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just
 115867 need to know that the asynchronous job failed; it does not matter whether it detected an error
 115868 and failed or was killed and did not complete its job normally.

115869 EXAMPLES

115870 Although the exact value used when a process is terminated by a signal is unspecified, if it is
 115871 known that a signal terminated a process, a script can still reliably determine which signal by
 115872 using *kill* as shown by the following script:

```
115873 sleep 1000&
115874 pid=$!
115875 kill -kill $pid
115876 wait $pid
115877 echo $pid was terminated by a SIG$(kill -l $?) signal.
```

115878 If the following sequence of commands is run in less than 31 seconds:

```
115879 sleep 257 | sleep 31 &
115880 jobs -l %%
```

115881 either of the following commands returns the exit status of the second *sleep* in the pipeline:

```
115882 wait <pid of sleep 31>
115883 wait %%
```

115884 RATIONALE

115885 The description of *wait* does not refer to the *waitpid()* function from the System Interfaces
 115886 volume of POSIX.1-2017 because that would needlessly overspecify this interface. However, the
 115887 wording means that *wait* is required to wait for an explicit process when it is given an argument
 115888 so that the status information of other processes is not consumed. Historical implementations
 115889 use the *wait()* function defined in the System Interfaces volume of POSIX.1-2017 until *wait()*
 115890 returns the requested process ID or finds that the requested process does not exist. Because this

115891 means that a shell script could not reliably get the status of all background children if a second
115892 background job was ever started before the first job finished, it is recommended that the *wait*
115893 utility use a method such as the functionality provided by the *waitpid()* function.

115894 The ability to wait for multiple *pid* operands was adopted from the KornShell.

115895 This new functionality was added because it is needed to determine the exit status of any
115896 asynchronous list accurately. The only compatibility problem that this change creates is for a
115897 script like

```
115898 while sleep 60 do  
115899     job& echo Job started $(date) as $! done
```

115900 which causes the shell to monitor all of the jobs started until the script terminates or runs out of
115901 memory. This would not be a problem if the loop did not reference "\$!" or if the script would
115902 occasionally *wait* for jobs it started.

115903 **FUTURE DIRECTIONS**

115904 None.

115905 **SEE ALSO**

115906 [Chapter 2](#) (on page 2345), *kill*, *sh*

115907 [XBD Section 3.204](#) (on page 66), [Chapter 8](#) (on page 173)

115908 XSH *wait()*

115909 **CHANGE HISTORY**

115910 First released in Issue 2.

115911 **NAME**115912 `wc` — word, line, and byte or character count115913 **SYNOPSIS**115914 `wc [-c|-m] [-lw] [file...]`115915 **DESCRIPTION**115916 The *wc* utility shall read one or more input files and, by default, write the number of <newline>
115917 characters, words, and bytes contained in each input file to the standard output.115918 The utility also shall write a total count for all named files, if more than one input file is
115919 specified.115920 The *wc* utility shall consider a *word* to be a non-zero-length string of characters delimited by
115921 white space.115922 **OPTIONS**115923 The *wc* utility shall conform to XBD [Section 12.2](#) (on page 216).

115924 The following options shall be supported:

115925 `-c` Write to the standard output the number of bytes in each input file.115926 `-l` Write to the standard output the number of <newline> characters in each input
115927 file.115928 `-m` Write to the standard output the number of characters in each input file.115929 `-w` Write to the standard output the number of words in each input file.115930 When any option is specified, *wc* shall report only the information requested by the specified
115931 options.115932 **OPERANDS**

115933 The following operand shall be supported:

115934 *file* A pathname of an input file. If no *file* operands are specified, the standard input
115935 shall be used.115936 **STDIN**115937 The standard input shall be used if no *file* operands are specified, and shall be used if a *file*
115938 operand is '-' and the implementation treats the '-' as meaning standard input. Otherwise,
115939 the standard input shall not be used. See the INPUT FILES section.115940 **INPUT FILES**

115941 The input files may be of any type.

115942 **ENVIRONMENT VARIABLES**115943 The following environment variables shall affect the execution of *wc*:115944 *LANG* Provide a default value for the internationalization variables that are unset or null.
115945 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
115946 variables used to determine the values of locale categories.)115947 *LC_ALL* If set to a non-empty string value, override the values of all the other
115948 internationalization variables.115949 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
115950 characters (for example, single-byte as opposed to multi-byte characters in
115951 arguments and input files) and which characters are defined as white-space
115952 characters.

- 115953 *LC_MESSAGES*
- 115954 Determine the locale that should be used to affect the format and contents of
- 115955 diagnostic messages written to standard error and informative messages written to
- 115956 standard output.
- 115957 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 115958 **ASYNCHRONOUS EVENTS**
- 115959 Default.
- 115960 **STDOUT**
- 115961 By default, the standard output shall contain an entry for each input file of the form:
- 115962 "%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>
- 115963 If the **-m** option is specified, the number of characters shall replace the <bytes> field in this
- 115964 format.
- 115965 If any options are specified and the **-l** option is not specified, the number of <newline>
- 115966 characters shall not be written.
- 115967 If any options are specified and the **-w** option is not specified, the number of words shall not be
- 115968 written.
- 115969 If any options are specified and neither **-c** nor **-m** is specified, the number of bytes or characters
- 115970 shall not be written.
- 115971 If no input *file* operands are specified, no name shall be written and no <blank> characters
- 115972 preceding the pathname shall be written.
- 115973 If more than one input *file* operand is specified, an additional line shall be written, of the same
- 115974 format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead
- 115975 of a pathname and the total of each column shall be written as appropriate. Such an additional
- 115976 line, if any, is written at the end of the output.
- 115977 **STDERR**
- 115978 The standard error shall be used only for diagnostic messages.
- 115979 **OUTPUT FILES**
- 115980 None.
- 115981 **EXTENDED DESCRIPTION**
- 115982 None.
- 115983 **EXIT STATUS**
- 115984 The following exit values shall be returned:
- 115985 0 Successful completion.
- 115986 >0 An error occurred.
- 115987 **CONSEQUENCES OF ERRORS**
- 115988 Default.

115989 **APPLICATION USAGE**

115990 The `-m` option is not a switch, but an option at the same level as `-c`. Thus, to produce the full
 115991 default output with character counts instead of bytes, the command required is:

115992 `wc -mlw`

115993 **EXAMPLES**

115994 None.

115995 **RATIONALE**

115996 The output file format pseudo-*printf()* string differs from the System V version of *wc*:

115997 `"%7d%7d%7d %s\n"`

115998 which produces possibly ambiguous and unparseable results for very large files, as it assumes no
 115999 number shall exceed six digits.

116000 Some historical implementations use only `<space>`, `<tab>`, and `<newline>` as word separators.
 116001 The equivalent of the ISO C standard *isspace()* function is more appropriate.

116002 The `-c` option stands for “character” count, even though it counts bytes. This stems from the
 116003 sometimes erroneous historical view that bytes and characters are the same size. Due to
 116004 international requirements, the `-m` option (reminiscent of “multi-byte”) was added to obtain
 116005 actual character counts.

116006 Early proposals only specified the results when input files were text files. The current
 116007 specification more closely matches historical practice. (Bytes, words, and `<newline>` characters
 116008 are counted separately and the results are written when an end-of-file is detected.)

116009 Historical implementations of the *wc* utility only accepted one argument to specify the options
 116010 `-c`, `-l`, and `-w`. Some of them also had multiple occurrences of an option cause the
 116011 corresponding count to be written multiple times and had the order of specification of the
 116012 options affect the order of the fields on output, but did not document either of these. Because
 116013 common usage either specifies no options or only one option, and because none of this was
 116014 documented, the changes required by this volume of POSIX.1-2017 should not break many
 116015 historical applications (and do not break any historical conforming applications).

116016 **FUTURE DIRECTIONS**

116017 None.

116018 **SEE ALSO**

116019 [*cksum*](#)

116020 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

116021 **CHANGE HISTORY**

116022 First released in Issue 2.

116023 **Issue 7**

116024 Austin Group Interpretation 1003.1-2001 #092 is applied.

116025 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116026 **NAME**116027 what `‡identify SCCS files(DEVELOPMENT)`116028 **SYNOPSIS**116029 XSI `what [-s] file...`116030 **DESCRIPTION**

116031 The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get*)
 116032 substitutes for the `%Z%` keyword ("`@(#)`") and shall write to standard output what follows
 116033 until the first occurrence of one of the following:

116034 " > newline \ NUL

116035 **OPTIONS**116036 The *what* utility shall conform to XBD [Section 12.2](#) (on page 216).

116037 The following option shall be supported:

116038 `-s` Quit after finding the first occurrence of the pattern in each file.116039 **OPERANDS**

116040 The following operands shall be supported:

116041 *file* A pathname of a file to search.116042 **STDIN**

116043 Not used.

116044 **INPUT FILES**

116045 The input files shall be of any file type.

116046 **ENVIRONMENT VARIABLES**116047 The following environment variables shall affect the execution of *what*:

116048 `LANG` Provide a default value for the internationalization variables that are unset or null.
 116049 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 116050 variables used to determine the values of locale categories.)

116051 `LC_ALL` If set to a non-empty string value, override the values of all the other
 116052 internationalization variables.

116053 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
 116054 characters (for example, single-byte as opposed to multi-byte characters in
 116055 arguments and input files).

116056 `LC_MESSAGES`

116057 Determine the locale that should be used to affect the format and contents of
 116058 diagnostic messages written to standard error.

116059 `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

116060 **ASYNCHRONOUS EVENTS**

116061 Default.

116062 **STDOUT**116063 The standard output shall consist of the following for each *file* operand:116064 "`%s:\n\t%s\n`", *<pathname>*, *<identification string>*

116065 **STDERR**

116066 The standard error shall be used only for diagnostic messages.

116067 **OUTPUT FILES**

116068 None.

116069 **EXTENDED DESCRIPTION**

116070 None.

116071 **EXIT STATUS**

116072 The following exit values shall be returned:

116073 0 Any matches were found.

116074 1 Otherwise.

116075 **CONSEQUENCES OF ERRORS**

116076 Default.

116077 **APPLICATION USAGE**

116078 The *what* utility is intended to be used in conjunction with the SCCS command *get*, which automatically inserts identifying information, but it can also be used where the information is inserted by any other means.

116081 When the string "@(#)" is included in a library routine in a shared library, it might not be found in an **a.out** file using that library routine.

116083 **EXAMPLES**

116084 If the C-language program in file **f.c** contains:

```
116085 char ident[] = "@(#)identification information";
```

116086 and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

```
116087 what f.c f.o a.out
```

116088 writes:

```
116089 f.c:
116090     identification information
116091     ...
116092 f.o:
116093     identification information
116094     ...
116095 a.out:
116096     identification information
116097     ...
```

116098 **RATIONALE**

116099 None.

116100 **FUTURE DIRECTIONS**

116101 None.

116102 **SEE ALSO**

116103 *get*

116104 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

116105 **CHANGE HISTORY**
116106 First released in Issue 2.

116107 **NAME**116108 who \ddagger 'display who is on the system116109 **SYNOPSIS**116110 XSI who [-mTu] [-abdHlprt] [*file*]116111 XSI who [-mu] -s [-bHlprt] [*file*]116112 who -q [*file*]

116113 who am i

116114 who am I

116115 **DESCRIPTION**116116 The *who* utility shall list various pieces of information about accessible users. The domain of
116117 accessibility is implementation-defined.116118 XSI Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed
116119 time since activity occurred on the line, and the process ID of the command interpreter for each
116120 current system user.116121 **OPTIONS**116122 The *who* utility shall conform to XBD [Section 12.2](#) (on page 216).116123 The following options shall be supported. The metavariables, such as *<line>*, refer to fields
116124 described in the STDOUT section.116125 XSI **-a** Process the implementation-defined database or named file with the **-b**, **-d**, **-l**, **-p**,
116126 **-r**, **-t**, **-T** and **-u** options turned on.116127 XSI **-b** Write the time and date of the last system reboot. The system reboot time is the
116128 time at which the implementation is able to commence running processes.116129 XSI **-d** Write a list of all processes that have expired and not been respawned by the *init*
116130 system process. The *<exit>* field shall appear for dead processes and contain the
116131 termination and exit values of the dead process. This can be useful in determining
116132 why a process terminated.116133 XSI **-H** Write column headings above the regular output.116134 XSI **-l** (The letter ell.) List only those lines on which the system is waiting for someone to
116135 login. The *<name>* field shall be **LOGIN** in such cases. Other fields shall be the
116136 same as for user entries except that the *<state>* field does not exist.116137 **-m** Output only information about the current terminal.116138 XSI **-p** List any other process that is currently active and has been previously spawned by
116139 *init*.116140 XSI **-q** (Quick.) List only the names and the number of users currently logged on. When
116141 this option is used, all other options shall be ignored.116142 XSI **-r** Write the current *run-level* of the *init* process.116143 XSI **-s** List only the *<name>*, *<line>*, and *<time>* fields. This is the default case.116144 XSI **-t** Indicate the last change to the system clock.116145 **-T** Show the state of each terminal, as described in the STDOUT section.

116146 **-u** Write ``idle time'' for each displayed user in addition to any other information. The
 116147 idle time is the time since any activity occurred on the user's terminal. The method
 116148 XSI of determining this is unspecified. This option shall list only those users who are
 116149 currently logged in. The *<name>* is the user's login name. The *<line>* is the name
 116150 of the line as found in the directory */dev*. The *<time>* is the time that the user
 116151 logged in. The *<activity>* is the number of hours and minutes since activity last
 116152 occurred on that particular line. A dot indicates that the terminal has seen activity
 116153 in the last minute and is therefore ``current''. If more than twenty-four hours have
 116154 elapsed or the line has not been used since boot time, the entry shall be marked
 116155 *<old>*. This field is useful when trying to determine whether a person is working at
 116156 the terminal or not. The *<pid>* is the process ID of the user's login process.

116157 OPERANDS

116158 XSI The following operands shall be supported:

116159 **am i, am I** In the POSIX locale, limit the output to describing the invoking user, equivalent to
 116160 the **-m** option. The **am** and **i** or **I** must be separate arguments.

116161 *file* Specify a pathname of a file to substitute for the implementation-defined database
 116162 of logged-on users that *who* uses by default.

116163 STDIN

116164 Not used.

116165 INPUT FILES

116166 None.

116167 ENVIRONMENT VARIABLES

116168 The following environment variables shall affect the execution of *who*:

116169 **LANG** Provide a default value for the internationalization variables that are unset or null.
 116170 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 116171 variables used to determine the values of locale categories.)

116172 **LC_ALL** If set to a non-empty string value, override the values of all the other
 116173 internationalization variables.

116174 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 116175 characters (for example, single-byte as opposed to multi-byte characters in
 116176 arguments).

116177 **LC_MESSAGES**

116178 Determine the locale that should be used to affect the format and contents of
 116179 diagnostic messages written to standard error.

116180 **LC_TIME** Determine the locale used for the format and contents of the date and time strings.

116181 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

116182 **TZ** Determine the timezone used when writing date and time information. If **TZ** is
 116183 unset or null, an unspecified default timezone shall be used.

116184 ASYNCHRONOUS EVENTS

116185 Default.

116186 STDOUT

116187 The *who* utility shall write its default format to the standard output in an implementation-
 116188 defined format, subject only to the requirement of containing the information described above.

116189 XSI OF XSI-conformant systems shall write the default information to the standard output in the
 116190 following general format:

```
116191 <name>[<state>]<line><time>[<activity>][<pid>][<comment>][<exit>]
```

116192 For the **-b** option, *<line>* shall be "system boot". The *<name>* is unspecified.

116193 The following format shall be used for the **-T** option:

```
116194 "%s %c %s %s\n" <name>, <terminal state>, <terminal name>,  
116195 <time of login>
```

116196 where *<terminal state>* is one of the following characters:

116197 + The terminal allows write access to other users.

116198 - The terminal denies write access to other users.

116199 ? The terminal write-access state cannot be determined.

116200 <space> This entry is not associated with a terminal.

116201 In the POSIX locale, the *<time of login>* shall be equivalent in format to the output of:

```
116202 date +"%b %e %H:%M"
```

116203 If the **-u** option is used with **-T**, the idle time shall be added to the end of the previous format in
 116204 an unspecified format.

116205 **STDERR**

116206 The standard error shall be used only for diagnostic messages.

116207 **OUTPUT FILES**

116208 None.

116209 **EXTENDED DESCRIPTION**

116210 None.

116211 **EXIT STATUS**

116212 The following exit values shall be returned:

116213 0 Successful completion.

116214 >0 An error occurred.

116215 **CONSEQUENCES OF ERRORS**

116216 Default.

116217 **APPLICATION USAGE**

116218 The name *init* used for the system process is the most commonly used on historical systems, but
 116219 it may vary.

116220 The “domain of accessibility” referred to is a broad concept that permits interpretation either on
 116221 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

116222 **EXAMPLES**

116223 None.

116224 **RATIONALE**

116225 Due to differences between historical implementations, the base options provided were a
 116226 compromise to allow users to work with those functions. The standard developers also
 116227 considered removing all the options, but felt that these options offered users valuable
 116228 functionality. Additional options to match historical systems are available on XSI-conformant

- 116229 systems.
- 116230 It is recognized that the *who* command may be of limited usefulness, especially in a multi-level
116231 secure environment. The standard developers considered, however, that having some standard
116232 method of determining the “accessibility” of other users would aid user portability.
- 116233 No format was specified for the default *who* output for systems not supporting the XSI option. In
116234 such a user-oriented command, designed only for human use, this was not considered to be a
116235 deficiency.
- 116236 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require
116237 that they use the same format.
- 116238 It is acceptable for an implementation to produce no output for an invocation of *who mil*.
- 116239 **FUTURE DIRECTIONS**
- 116240 None.
- 116241 **SEE ALSO**
- 116242 *mesg*
- 116243 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)
- 116244 **CHANGE HISTORY**
- 116245 First released in Issue 2.
- 116246 **Issue 6**
- 116247 This utility is marked as part of the User Portability Utilities option.
- 116248 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.
- 116249 **Issue 7**
- 116250 SD5-XCU-ERN-58 is applied, clarifying the **-b** option.
- 116251 The *who* utility is moved from the User Portability Utilities option to the Base. User Portability
116252 Utilities is now an option for interactive utilities.
- 116253 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116254 **NAME**

116255 write ‡write to another user

116256 **SYNOPSIS**116257 write *user_name* [*terminal*]116258 **DESCRIPTION**116259 The *write* utility shall read lines from the standard input and write them to the terminal of the
116260 specified user. When first invoked, it shall write the message:116261 **Message from** *sender-login-id* (*sending-terminal*) [*date*]...116262 to *user_name*. When it has successfully completed the connection, the sender's terminal shall be
116263 alerted twice to indicate that what the sender is typing is being written to the recipient's
116264 terminal.

116265 If the recipient wants to reply, this can be accomplished by typing:

116266 write *sender-login-id* [*sending-terminal*]116267 upon receipt of the initial message. Whenever a line of input as delimited by an NL, EOF, or
116268 EOL special character (see XBD [Chapter 11](#), on page 199) is accumulated while in canonical
116269 input mode, the accumulated data shall be written on the other user's terminal. Characters shall
116270 be processed as follows:

116271 Typing <alert> shall write the <alert> character to the recipient's terminal.

116272 Typing the erase and kill characters shall affect the sender's terminal in the manner
116273 described by the **termios** interface in XBD [Chapter 11](#) (on page 199).116274 Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate
116275 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.116276 Typing characters from *LC_CTYPE* classifications **print** or **space** shall cause those
116277 characters to be sent to the recipient's terminal.116278 When and only when the *stty* **ixten** local mode is enabled, the existence and processing of
116279 additional special control characters and multi-byte or single-byte functions is
116280 implementation-defined.116281 Typing other non-printable characters shall cause implementation-defined sequences of
116282 printable characters to be written to the recipient's terminal.116283 To write to a user who is logged in more than once, the *terminal* argument can be used to
116284 indicate which terminal to write to; otherwise, the recipient's terminal is selected in an
116285 implementation-defined manner and an informational message is written to the sender's
116286 standard output, indicating which terminal was chosen.116287 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*
116288 utility. However, a user's privilege may further constrain the domain of accessibility of other
116289 users' terminals. The *write* utility shall fail when the user lacks appropriate privileges to perform
116290 the requested action.116291 **OPTIONS**

116292 None.

116293 **OPERANDS**

- 116294 The following operands shall be supported:
- 116295 *user_name* Login name of the person to whom the message shall be written. The application
116296 shall ensure that this operand is of the form returned by the *who* utility.
- 116297 *terminal* Terminal identification in the same format provided by the *who* utility.
- 116298 **STDIN**
- 116299 Lines to be copied to the recipient's terminal are read from standard input.
- 116300 **INPUT FILES**
- 116301 None.
- 116302 **ENVIRONMENT VARIABLES**
- 116303 The following environment variables shall affect the execution of *write*:
- 116304 *LANG* Provide a default value for the internationalization variables that are unset or null.
116305 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
116306 variables used to determine the values of locale categories.)
- 116307 *LC_ALL* If set to a non-empty string value, override the values of all the other
116308 internationalization variables.
- 116309 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
116310 characters (for example, single-byte as opposed to multi-byte characters in
116311 arguments and input files). If the recipient's locale does not use an *LC_CTYPE*
116312 equivalent to the sender's, the results are undefined.
- 116313 *LC_MESSAGES*
- 116314 Determine the locale that should be used to affect the format and contents of
116315 diagnostic messages written to standard error and informative messages written to
116316 standard output.
- 116317 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 116318 **ASYNCHRONOUS EVENTS**
- 116319 If an interrupt signal is received, *write* shall write an appropriate message on the recipient's
116320 terminal and exit with a status of zero. It shall take the standard action for all other signals.
- 116321 **STDOUT**
- 116322 An informational message shall be written to standard output if a recipient is logged in more
116323 than once.
- 116324 **STDERR**
- 116325 The standard error shall be used only for diagnostic messages.
- 116326 **OUTPUT FILES**
- 116327 The recipient's terminal is used for output.
- 116328 **EXTENDED DESCRIPTION**
- 116329 None.
- 116330 **EXIT STATUS**
- 116331 The following exit values shall be returned:
- 116332 0 Successful completion.
- 116333 >0 The addressed user is not logged on or the addressed user denies permission.

116334 **CONSEQUENCES OF ERRORS**

116335 Default.

116336 **APPLICATION USAGE**116337 The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.116338 **EXAMPLES**

116339 None.

116340 **RATIONALE**

116341 The *write* utility was included in this volume of POSIX.1-2017 since it can be implemented on all
116342 terminal types. The standard developers considered the *talk* utility, which cannot be
116343 implemented on certain terminals, to be a “better” communications interface. Both of these
116344 programs are in widespread use on historical implementations. Therefore, the standard
116345 developers decided that both utilities should be specified.

116346 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
116347 require that they all use or accept the same format.

116348 **FUTURE DIRECTIONS**

116349 None.

116350 **SEE ALSO**116351 *mesg*, *talk*, *who*116352 XBD [Chapter 8](#) (on page 173), [Chapter 11](#) (on page 199)116353 **CHANGE HISTORY**

116354 First released in Issue 2.

116355 **Issue 5**

116356 The FUTURE DIRECTIONS section is added.

116357 **Issue 6**

116358 This utility is marked as part of the User Portability Utilities option.

116359 The normative text is reworded to avoid use of the term “must” for application requirements.

116360 **Issue 7**116361 The *write* utility is moved from the User Portability Utilities option to the Base. User Portability
116362 Utilities is now an option for interactive utilities.

116363 **NAME**116364 `xargs` — construct argument lists and invoke utility116365 **SYNOPSIS**

```
116366 XSI xargs [-ptx] [-E eofstr] [-I replstr|-L number|-n number]
116367 [-s size] [utility [argument...]]
```

116368 **DESCRIPTION**

116369 The `xargs` utility shall construct a command line consisting of the *utility* and *argument* operands
 116370 specified followed by as many arguments read in sequence from standard input as fit in length
 116371 and number constraints specified by the options. The `xargs` utility shall then invoke the
 116372 constructed command line and wait for its completion. This sequence shall be repeated until one
 116373 of the following occurs:

116374 An end-of-file condition is detected on standard input.

116375 An argument consisting of just the logical end-of-file string (see the `-E eofstr` option) is
 116376 found on standard input after double-quote processing, <apostrophe> processing, and
 116377 <backslash>-escape processing (see next paragraph). All arguments up to but not
 116378 including the argument consisting of just the logical end-of-file string shall be used as
 116379 arguments in constructed command lines.

116380 An invocation of a constructed command line returns an exit status of 255.

116381 The application shall ensure that arguments in the standard input are separated by unquoted
 116382 <blank> characters, unescaped <blank> characters, or <newline> characters. A string of zero or
 116383 more non-double-quote ('"') characters and non-<newline> characters can be quoted by
 116384 enclosing them in double-quotes. A string of zero or more non-<apostrophe> ('\ ') characters
 116385 and non-<newline> characters can be quoted by enclosing them in <apostrophe> characters.
 116386 Any unquoted character can be escaped by preceding it with a <backslash>. The utility named
 116387 by *utility* shall be executed one or more times until the end-of-file is reached or the logical end-of-
 116388 file string is found. The results are unspecified if the utility named by *utility* attempts to read
 116389 from its standard input.

116390 The generated command line length shall be the sum of the size in bytes of the utility name and
 116391 each argument treated as strings, including a null byte terminator for each of these strings. The
 116392 `xargs` utility shall limit the command line length such that when the command line is invoked,
 116393 the combined argument and environment lists (see the *exec* family of functions in the System
 116394 Interfaces volume of POSIX.1-2017) shall not exceed {ARG_MAX}-2048 bytes. Within this
 116395 constraint, if neither the `-n` nor the `-s` option is specified, the default command line length shall
 116396 be at least {LINE_MAX}.

116397 **OPTIONS**

116398 The `xargs` utility shall conform to XBD [Section 12.2](#) (on page 216).

116399 The following options shall be supported:

116400 `-E eofstr` Use *eofstr* as the logical end-of-file string. If `-E` is not specified, it is unspecified
 116401 whether the logical end-of-file string is the <underscore> character ('_') or the
 116402 end-of-file string capability is disabled. When *eofstr* is the null string, the logical
 116403 end-of-file string capability shall be disabled and <underscore> characters shall be
 116404 taken literally.

116405 XSI `-I replstr` Insert mode: *utility* is executed for each logical line from standard input.
 116406 Arguments in the standard input shall be separated only by unescaped <newline>
 116407 characters, not by <blank> characters. Any unquoted unescaped <blank>
 116408 characters at the beginning of each line shall be ignored. The resulting argument
 116409 shall be inserted in *arguments* in place of each occurrence of *replstr*. At least five

- 116410 arguments in *arguments* can each contain one or more instances of *replstr*. Each of
 116411 these constructed arguments cannot grow larger than an implementation-defined
 116412 limit greater than or equal to 255 bytes. Option *-x* shall be forced on.
- 116413 XSI *-L number* The *utility* shall be executed for each non-empty *number* lines of arguments from
 116414 standard input. The last invocation of *utility* shall be with fewer lines of arguments
 116415 if fewer than *number* remain. A line is considered to end with the first <newline>
 116416 unless the last character of the line is an unescaped <blank>; a trailing unescaped
 116417 <blank> signals continuation to the next non-empty line, inclusive.
- 116418 *-n number* Invoke *utility* using as many standard input arguments as possible, up to *number* (a
 116419 positive decimal integer) arguments maximum. Fewer arguments shall be used if:
- 116420 The command line length accumulated exceeds the size specified by the *-s*
 116421 option (or {LINE_MAX} if there is no *-s* option).
- 116422 The last iteration has fewer than *number*, but not zero, operands remaining.
- 116423 *-p* Prompt mode: the user is asked whether to execute *utility* at each invocation. Trace
 116424 mode (*-t*) is turned on to write the command instance to be executed, followed by
 116425 a prompt to standard error. An affirmative response read from */dev/tty* shall
 116426 execute the command; otherwise, that particular invocation of *utility* shall be
 116427 skipped.
- 116428 *-s size* Invoke *utility* using as many standard input arguments as possible yielding a
 116429 command line length less than *size* (a positive decimal integer) bytes. Fewer
 116430 arguments shall be used if:
- 116431 The total number of arguments exceeds that specified by the *-n* option.
- 116432 XSI The total number of lines exceeds that specified by the *-L* option.
- 116433 End-of-file is encountered on standard input before *size* bytes are
 116434 accumulated.
- 116435 Values of *size* up to at least {LINE_MAX} bytes shall be supported, provided that
 116436 the constraints specified in the DESCRIPTION are met. It shall not be considered
 116437 an error if a value larger than that supported by the implementation or exceeding
 116438 the constraints specified in the DESCRIPTION is given; *xargs* shall use the largest
 116439 value it supports within the constraints.
- 116440 *-t* Enable trace mode. Each generated command line shall be written to standard
 116441 error just prior to invocation.
- 116442 *-x* Terminate if a constructed command line will not fit in the implied or specified size
 116443 (see the *-s* option above).

116444 OPERANDS

- 116445 The following operands shall be supported:
- 116446 *utility* The name of the utility to be invoked, found by search path using the *PATH*
 116447 environment variable, described in XBD [Chapter 8](#) (on page 173). If *utility* is
 116448 omitted, the default shall be the *echo* utility. If the *utility* operand names any of the
 116449 special built-in utilities in [Section 2.14](#) (on page 2384), the results are undefined.
- 116450 *argument* An initial option or operand for the invocation of *utility*.

116451 **STDIN**

116452 The standard input shall be a text file. The results are unspecified if an end-of-file condition is
 116453 detected immediately following an escaped <newline>.

116454 **INPUT FILES**

116455 The file `/dev/tty` shall be used to read responses required by the `-p` option.

116456 **ENVIRONMENT VARIABLES**

116457 The following environment variables shall affect the execution of *xargs*:

116458 *LANG* Provide a default value for the internationalization variables that are unset or null.
 116459 (See XBD Section 8.2 (on page 174) for the precedence of internationalization
 116460 variables used to determine the values of locale categories.)

116461 *LC_ALL* If set to a non-empty string value, override the values of all the other
 116462 internationalization variables.

116463 *LC_COLLATE*

116464 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 116465 character collating elements used in the extended regular expression defined for
 116466 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

116467 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 116468 characters (for example, single-byte as opposed to multi-byte characters in
 116469 arguments and input files) and the behavior of character classes used in the
 116470 extended regular expression defined for the **yesexpr** locale keyword in the
 116471 *LC_MESSAGES* category.

116472 *LC_MESSAGES*

116473 Determine the locale used to process affirmative responses, and the locale used to
 116474 affect the format and contents of diagnostic messages and prompts written to
 116475 standard error.

116476 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

116477 *PATH* Determine the location of *utility*, as described in XBD Chapter 8 (on page 173).

116478 **ASYNCHRONOUS EVENTS**

116479 Default.

116480 **STDOUT**

116481 Not used.

116482 **STDERR**

116483 The standard error shall be used for diagnostic messages and the `-t` and `-p` options. If the `-t`
 116484 option is specified, the *utility* and its constructed argument list shall be written to standard error,
 116485 as it will be invoked, prior to invocation. If `-p` is specified, a prompt of the following format
 116486 shall be written (in the POSIX locale):

116487 " ? . . . "

116488 at the end of the line of the output from `-t`.

116489 **OUTPUT FILES**

116490 None.

116491 **EXTENDED DESCRIPTION**

116492 None.

116493 **EXIT STATUS**

116494 The following exit values shall be returned:

- 116495 0 All invocations of *utility* returned exit status zero.
- 116496 1-125 A command line meeting the specified requirements could not be assembled, one or more of the invocations of *utility* returned a non-zero exit status, or some other error occurred.
- 116497
- 116498
- 116499 126 The utility specified by *utility* was found but could not be invoked.
- 116500 127 The utility specified by *utility* could not be found.

116501 **CONSEQUENCES OF ERRORS**

116502 If a command line meeting the specified requirements cannot be assembled, the utility cannot be
 116503 invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits
 116504 with exit status 255, the *xargs* utility shall write a diagnostic message and exit without
 116505 processing any remaining input.

116506 **APPLICATION USAGE**

116507 The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no
 116508 further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit*
 116509 with an appropriate value to avoid accidentally returning with 255.

116510 Note that since input is parsed as lines, <blank> characters separate arguments, and
 116511 <backslash>, <apostrophe>, and double-quote characters are used for quoting, if *xargs* is used to
 116512 bundle the output of commands like *find dir -print* or *ls* into commands to be executed,
 116513 unexpected results are likely if any filenames contain <blank>, <newline>, or quoting characters.
 116514 This can be solved by using *find* to call a script that converts each file found into a quoted string
 116515 that is then piped to *xargs*, but in most cases it is preferable just to have *find* do the argument
 116516 aggregation itself by using *-exec* with a '+' terminator instead of ';' . Note that the quoting
 116517 rules used by *xargs* are not the same as in the shell. They were not made consistent here because
 116518 existing applications depend on the current rules. An easy (but inefficient) method that can be
 116519 used to transform input consisting of one argument per line into a quoted form that *xargs*
 116520 interprets correctly is to precede each non-<newline> character with a <backslash>. More
 116521 efficient alternatives are shown in Example 2 and Example 5 below.

116522 On implementations with a large value for {ARG_MAX}, *xargs* may produce command lines
 116523 longer than {LINE_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used
 116524 to create a text file, users should explicitly set the maximum command line length with the *-s*
 116525 option.

116526 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
 116527 an error occurs so that applications can distinguish "failure to find a utility" from "invoked
 116528 utility exited with an error indication". The value 127 was chosen because it is not commonly
 116529 used for other meanings; most utilities use small values for "normal error conditions" and the
 116530 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
 116531 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
 116532 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
 116533 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
 116534 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
 116535 any other reason.

116536 EXAMPLES

116537 1. The following command combines the output of the parenthesized commands (minus the
116538 <apostrophe> characters) onto one line, which is then appended to the file log. It assumes
116539 that the expansion of "\$0 \$*" does not include any <apostrophe> or <newline>
116540 characters.

```
116541 (logname; date; printf "'%s'\n" "$0 $*") | xargs -E "" >>log
```

116542 2. The following command invokes *diff* with successive pairs of arguments originally typed
116543 as command line arguments. It assumes there are no embedded <newline> characters in
116544 the elements of the original argument list.

```
116545 printf "%s\n" "$@" | sed 's/^[[:alnum:]]/\\&/g' |  
116546 xargs -E "" -n 2 -x diff
```

116547 3. In the following commands, the user is asked which files in the current directory
116548 (excluding dotfiles) are to be archived. The files are archived into **arch**; *a*, one at a time or
116549 *b*, many at a time. The commands assume that no filenames contain <blank>, <newline>,
116550 <backslash>, <apostrophe>, or double-quote characters.

```
116551 a. ls | xargs -E "" -p -L 1 ar -r arch
```

```
116552 b. ls | xargs -E "" -p -L 1 | xargs -E "" ar -r arch
```

116553 4. The following command invokes *command1* one or more times with multiple arguments,
116554 stopping if an invocation of *command1* has a non-zero exit status.

```
116555 xargs -E "" sh -c 'command1 "$@" || exit 255' sh < xargs_input
```

116556 5. On XSI-conformant systems, the following command moves all files from directory **\$1** to
116557 directory **\$2**, and echoes each move command just before doing it. It assumes no
116558 filenames contain <newline> characters and that neither **\$1** nor **\$2** contains the sequence
116559 "{}".

```
116560 ls -A "$1" | sed -e 's/"\/"\/"/g' -e 's/.*/"&"/' |  
116561 xargs -E "" -I {} -t mv "$1"/{} "$2"/{}
```

116562 RATIONALE

116563 The *xargs* utility was usually found only in System V-based systems; BSD systems included an
116564 *apply* utility that provided functionality similar to *xargs -n number*. The SVID lists *xargs* as a
116565 software development extension. This volume of POSIX.1-2017 does not share the view that it is
116566 used only for development, and therefore it is not optional.

116567 The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the
116568 number of processes launched by a simplistic use of the *find -exec* combination. The *xargs* utility
116569 is also used to enforce an upper limit on memory required to launch a process. With this basis in
116570 mind, this volume of POSIX.1-2017 selected only the minimal features required.

116571 Although the 255 exit status is mostly an accident of historical implementations, it allows a
116572 utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the
116573 current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125
116574 range when *xargs* exits. There is no statement of how the various non-zero utility exit status
116575 codes are accumulated by *xargs*. The value could be the addition of all codes, their highest
116576 value, the last one received, or a single value such as 1. Since no algorithm is arguably better
116577 than the others, and since many of the standard utilities say little more (portably) than
116578 “pass/fail”, no new algorithm was invented.

116579 Several other *xargs* options were removed because simple alternatives already exist within this

116580 volume of POSIX.1-2017. For example, the `-i replstr` option can be just as efficiently performed
 116581 using a shell `for` loop. Since `xargs` calls an `exec` function with each input line, the `-i` option does
 116582 not usually exploit the grouping capabilities of `xargs`.

116583 The requirement that `xargs` never produces command lines such that invocation of `utility` is
 116584 within 2048 bytes of hitting the POSIX `exec {ARG_MAX}` limitations is intended to guarantee
 116585 that the invoked utility has room to modify its environment variables and command line
 116586 arguments and still be able to invoke another utility. Note that the minimum `{ARG_MAX}`
 116587 allowed by the System Interfaces volume of POSIX.1-2017 is 4096 bytes and the minimum value
 116588 allowed by this volume of POSIX.1-2017 is 2048 bytes; therefore, the 2048 bytes difference seems
 116589 reasonable. Note, however, that `xargs` may never be able to invoke a utility if the environment
 116590 passed in to `xargs` comes close to using `{ARG_MAX}` bytes.

116591 The version of `xargs` required by this volume of POSIX.1-2017 is required to wait for the
 116592 completion of the invoked command before invoking another command. This was done because
 116593 historical scripts using `xargs` assumed sequential execution. Implementations wanting to provide
 116594 parallel operation of the invoked utilities are encouraged to add an option enabling parallel
 116595 invocation, but should still wait for termination of all of the children before `xargs` terminates
 116596 normally.

116597 The `-e` option was omitted from the ISO POSIX-2:1993 standard in the belief that the `eofstr`
 116598 option-argument was recognized only when it was on a line by itself and before quote and
 116599 escape processing were performed, and that the logical end-of-file processing was only enabled
 116600 if a `-e` option was specified. In that case, a simple `sed` script could be used to duplicate the `-e`
 116601 functionality. Further investigation revealed that:

116602 The logical end-of-file string was checked for after quote and escape processing, making a
 116603 `sed` script that provided equivalent functionality much more difficult to write.

116604 The default was to perform logical end-of-file processing with an `<underscore>` as the
 116605 logical end-of-file string.

116606 To correct this misunderstanding, the `-E eofstr` option was adopted from the X/Open Portability
 116607 Guide. Users should note that the description of the `-E` option matches historical documentation
 116608 of the `-e` option (which was not adopted because it did not support the Utility Syntax
 116609 Guidelines), by saying that if `eofstr` is the null string, logical end-of-file processing is disabled.
 116610 Historical implementations of `xargs` actually did not disable logical end-of-file processing; they
 116611 treated a null argument found in the input as a logical end-of-file string. (A null `string` argument
 116612 could be generated using single or double-quotes (' ' or " "). Since this behavior was not
 116613 documented historically, it is considered to be a bug.

116614 The `-I`, `-L`, and `-n` options are mutually-exclusive. Some implementations use the last one
 116615 specified if more than one is given on a command line; other implementations treat
 116616 combinations of the options in different ways.

116617 **FUTURE DIRECTIONS**

116618 None.

116619 **SEE ALSO**

116620 Chapter 2 (on page 2345), [diff](#), [echo](#), [find](#)

116621 XBD Chapter 8 (on page 173), [Section 12.2](#) (on page 216)

116622 XSH [exec](#)

116623 **CHANGE HISTORY**

116624 First released in Issue 2.

116625 **Issue 5**

116626 A second FUTURE DIRECTION is added.

116627 **Issue 6**116628 The obsolescent `-e`, `-i`, and `-l` options are removed.116629 The following new requirements on POSIX implementations derive from alignment with the
116630 Single UNIX Specification:116631 The `-p` option is added.116632 In the INPUT FILES section, the file `/dev/tty` is used to read responses required by the `-p`
116633 option.116634 The STDERR section is updated to describe the `-p` option.116635 The description of the `-E` option is aligned with the ISO POSIX-2: 1993 standard.

116636 The normative text is reworded to avoid use of the term “must” for application requirements.

116637 **Issue 7**116638 Austin Group Interpretation 1003.1-2001 #123 is applied, changing the description of the `xargs -I`
116639 option.116640 Austin Group Interpretation 1003.1-2001 #126 is applied, changing the description of the
116641 `LC_MESSAGES` environment variable.

116642 SD5-XCU-ERN-68 is applied.

116643 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

116644 SD5-XCU-ERN-128 is applied, clarifying the DESCRIPTION of the logical end-of-file string.

116645 SD5-XCU-ERN-132 is applied, updating the EXAMPLES section.

116646 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0149 [342] is applied.

116647 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0203 [499] is applied.

116648 **NAME**116649 yacc ‡yet another compiler compiler **DEVELOPMENT**)116650 **SYNOPSIS**116651 CD yacc [-dltv] [-b *file_prefix*] [-p *sym_prefix*] *grammar*116652 **DESCRIPTION**

116653 The *yacc* utility shall read a description of a context-free grammar in *grammar* and write C source
 116654 code, conforming to the ISO C standard, to a code file, and optionally header information into a
 116655 header file, in the current directory. The generated source code shall not depend on any
 116656 undefined, unspecified, or implementation-defined behavior, except in cases where it is copied
 116657 directly from the supplied grammar, or in cases that are documented by the implementation.
 116658 The C code shall define a function and related routines and macros for an automaton that
 116659 executes a parsing algorithm meeting the requirements in [Algorithms](#) (on page 3465).

116660 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

116661 The C source code and header file shall be produced in a form suitable as input for the C
 116662 compiler (see [c99](#)).

116663 **OPTIONS**

116664 The *yacc* utility shall conform to XBD [Section 12.2](#) (on page 216), except for Guideline 9.

116665 The following options shall be supported:

116666 **-b** *file_prefix* Use *file_prefix* instead of **y** as the prefix for all output filenames. The code file
 116667 **y.tab.c**, the header file **y.tab.h** (created when **-d** is specified), and the description
 116668 file **y.output** (created when **-v** is specified), shall be changed to *file_prefix.tab.c*,
 116669 *file_prefix.tab.h*, and *file_prefix.output*, respectively.

116670 **-d** Write the header file; by default only the code file is written. See the OUTPUT
 116671 FILES section.

116672 **-l** Produce a code file that does not contain any **#line** constructs. If this option is not
 116673 present, it is unspecified whether the code file or header file contains **#line**
 116674 directives. This should only be used after the grammar and the associated actions
 116675 are fully debugged.

116676 **-p** *sym_prefix*

116677 Use *sym_prefix* instead of **yy** as the prefix for all external names produced by *yacc*.
 116678 The names affected shall include the functions *yyparse()*, *yylex()*, and *yyerror()*,
 116679 and the variables *yyval*, *yychar*, and *yydebug*. (In the remainder of this section, the
 116680 six symbols cited are referenced using their default names only as a notational
 116681 convenience.) Local names may also be affected by the **-p** option; however, the **-p**
 116682 option shall not affect **#define** symbols generated by *yacc*.

116683 **-t** Modify conditional compilation directives to permit compilation of debugging
 116684 code in the code file. Runtime debugging statements shall always be contained in
 116685 the code file, but by default conditional compilation directives prevent their
 116686 compilation.

116687 **-v** Write a file containing a description of the parser and a report of conflicts
 116688 generated by ambiguities in the grammar.

116689 **OPERANDS**

116690 The following operand is required:

116691 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a
 116692 parser is to be created. The format for the grammar is described in the EXTENDED
 116693 DESCRIPTION section.

116694 **STDIN**

116695 Not used.

116696 **INPUT FILES**

116697 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION
 116698 section.

116699 **ENVIRONMENT VARIABLES**

116700 The following environment variables shall affect the execution of *yacc*:

116701 *LANG* Provide a default value for the internationalization variables that are unset or null.
 116702 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 116703 variables used to determine the values of locale categories.)

116704 *LC_ALL* If set to a non-empty string value, override the values of all the other
 116705 internationalization variables.

116706 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 116707 characters (for example, single-byte as opposed to multi-byte characters in
 116708 arguments and input files).

116709 *LC_MESSAGES*

116710 Determine the locale that should be used to affect the format and contents of
 116711 diagnostic messages written to standard error.

116712 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

116713 The *LANG* and *LC_** variables affect the execution of the *yacc* utility as stated. The *main()*
 116714 function defined in [Yacc Library](#) (on page 3465) shall call:

```
116715 setlocale(LC_ALL, "");
```

116716 and thus the program generated by *yacc* shall also be affected by the contents of these variables
 116717 at runtime.

116718 **ASYNCHRONOUS EVENTS**

116719 Default.

116720 **STDOUT**

116721 Not used.

116722 **STDERR**

116723 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of
 116724 those conflicts to the standard error in an unspecified format.

116725 Standard error shall also be used for diagnostic messages.

116726 **OUTPUT FILES**

116727 The code file, the header file, and the description file shall be text files. All are described in the
 116728 following sections.

116729 **Code File**

116730 This file shall contain the C source code for the *yyparse()* function. It shall contain code for the
 116731 various semantic actions with macro substitution performed on them as described in the
 116732 EXTENDED DESCRIPTION section. It also shall contain a copy of the **#define** statements in the
 116733 header file. If a **%union** declaration is used, the declaration for YYSTYPE shall also be included
 116734 in this file.

116735 **Header File**

116736 The header file shall contain **#define** statements that associate the token numbers with the token
 116737 names. This allows source files other than the code file to access the token codes. If a **%union**
 116738 declaration is used, the declaration for YYSTYPE and an *extern YYSTYPE yylval* declaration shall
 116739 also be included in this file.

116740 **Description File**

116741 The description file shall be a text file containing a description of the state machine
 116742 corresponding to the parser, using an unspecified format. Limits for internal tables (see [Limits](#),
 116743 on page 3466) shall also be reported, in an implementation-defined manner. (Some
 116744 implementations may use dynamic allocation techniques and have no specific limit values to
 116745 report.)

116746 **EXTENDED DESCRIPTION**

116747 The *yacc* command accepts a language that is used to define a grammar for a target language to
 116748 be parsed by the tables and code generated by *yacc*. The language accepted by *yacc* as a
 116749 grammar for the target language is described below using the *yacc* input language itself.

116750 The input *grammar* includes rules describing the input structure of the target language and code
 116751 to be invoked when these rules are recognized to provide the associated semantic action. The
 116752 code to be executed shall appear as bodies of text that are intended to be C-language code. These
 116753 bodies of text shall not contain C-language trigraphs. The C-language inclusions are presumed
 116754 to form a correct function when processed by *yacc* into its output files. The code included in this
 116755 way shall be executed during the recognition of the target language.

116756 Given a grammar, the *yacc* utility generates the files described in the OUTPUT FILES section.
 116757 The code file can be compiled and linked using *c99*. If the declaration and programs sections of
 116758 the grammar file did not include definitions of *main()*, *yylex()*, and *yyerror()*, the compiled
 116759 output requires linking with externally supplied versions of those functions. Default versions of
 116760 *main()* and *yyerror()* are supplied in the *yacc* library and can be linked in by using the *-ly*
 116761 operand to *c99*. The *yacc* library interfaces need not support interfaces with other than the
 116762 default **yy** symbol prefix. The application provides the lexical analyzer function, *yylex()*; the *lex*
 116763 utility is specifically designed to generate such a routine.

116764 **Input Language**

116765 The application shall ensure that every specification file consists of three sections in order:
 116766 *declarations*, *grammar rules*, and *programs*, separated by double <percent-sign> characters ("%").
 116767 The declarations and programs sections can be empty. If the latter is empty, the preceding "%"
 116768 mark separating it from the rules section can be omitted.

116769 The input is free form text following the structure of the grammar defined below.

116770 **Lexical Structure of the Grammar**

116771 The <blank>, <newline>, and <form-feed> character shall be ignored, except that the application
116772 shall ensure that they do not appear in names or multi-character reserved symbols. Comments
116773 shall be enclosed in `"/ * . . . */`, and can appear wherever a name is valid.

116774 Names are of arbitrary length, made up of letters, periods ('.'), underscores ('_'), and non-
116775 initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not
116776 use names beginning in **yy** or **YY** since the *yacc* parser uses such names. Many of the names
116777 appear in the final output of *yacc*, and thus they should be chosen to conform with any
116778 additional rules created by the C compiler to be used. In particular they appear in **#define**
116779 statements.

116780 A literal shall consist of a single character enclosed in single-quote characters. All of the escape
116781 sequences supported for character constants by the ISO C standard shall be supported by *yacc*.

116782 The relationship with the lexical analyzer is discussed in detail below.

116783 The application shall ensure that the NUL character is not used in grammar rules or literals.

116784 **Declarations Section**

116785 The declarations section is used to define the symbols used to define the target language and
116786 their relationship with each other. In particular, much of the additional information required to
116787 resolve ambiguities in the context-free grammar for the target language is provided here.

116788 Usually *yacc* assigns the relationship between the symbolic names it generates and their
116789 underlying numeric value. The declarations section makes it possible to control the assignment
116790 of these values.

116791 It is also possible to keep semantic information associated with the tokens currently on the parse
116792 stack in a user-defined C-language **union**, if the members of the union are associated with the
116793 various names in the grammar. The declarations section provides for this as well.

116794 The first group of declarators below all take a list of names as arguments. That list can optionally
116795 be preceded by the name of a C union member (called a *tag* below) appearing within '`<`' and
116796 '`>`'. (As an exception to the typographical conventions of the rest of this volume of
116797 POSIX.1-2017, in this case `<tag>` does not represent a metavariable, but the literal angle bracket
116798 characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this line
116799 shall be of the same C type as the union member referenced by *tag*. This is discussed in more
116800 detail below.

116801 For lists used to define tokens, the first appearance of a given token can be followed by a
116802 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it
116803 for lexical purposes shall be taken to be that number.

116804 The following declares *name* to be a token:

```
116805 %token [<tag>] name [number] [name [number]]...
```

116806 If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced
116807 by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

116808 The following declares *name* to be a token, and assigns precedence to it:

```
116809 %left [<tag>] name [number] [name [number]]...
```

```
116810 %right [<tag>] name [number] [name [number]]...
```

116811 One or more lines, each beginning with one of these symbols, can appear in this section. All
116812 tokens on the same line have the same precedence level and associativity; the lines are in order

116813 of increasing precedence or binding strength. **%left** denotes that the operators on that line are
 116814 left associative, and **%right** similarly denotes right associative operators. If *tag* is present, it shall
 116815 declare a C type for *names* as described for **%token**.

116816 The following declares *name* to be a token, and indicates that this cannot be used associatively:

```
116817 %nonassoc [<tag>] name [number] [name [number]]...
```

116818 If the parser encounters associative use of this token it reports an error. If *tag* is present, it shall
 116819 declare a C type for *names* as described for **%token**.

116820 The following declares that union member *names* are non-terminals, and thus it is required to
 116821 have a *tag* field at its beginning:

```
116822 %type <tag> name...
```

116823 Because it deals with non-terminals only, assigning a token number or using a literal is also
 116824 prohibited. If this construct is present, *yacc* shall perform type checking; if this construct is not
 116825 present, the parse stack shall hold only the **int** type.

116826 Every name used in *grammar* not defined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration
 116827 is assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any non-
 116828 terminal symbol that does not appear on the left side of at least one grammar rule.

116829 Once the type, precedence, or token number of a name is specified, it shall not be changed. If the
 116830 first declaration of a token does not assign a token number, *yacc* shall assign a token number.
 116831 Once this assignment is made, the token number shall not be changed by explicit assignment.

116832 The following declarators do not follow the previous pattern.

116833 The following declares the non-terminal *name* to be the *start symbol*, which represents the largest,
 116834 most general structure described by the grammar rules:

```
116835 %start name
```

116836 By default, it is the left-hand side of the first grammar rule; this default can be overridden with
 116837 this declaration.

116838 The following declares the *yacc* value stack to be a union of the various types of values desired.

```
116839 %union { body of union (in C) }
```

116840 The body of the union shall not contain unbalanced curly brace preprocessing tokens.

116841 By default, the values returned by actions (see below) and the lexical analyzer shall be of type
 116842 **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names
 116843 in order to perform strict type checking of the resulting parser.

116844 Alternatively, given that at least one *<tag>* construct is used, the union can be declared in a
 116845 header file (which shall be included in the declarations section by using a **#include** construct
 116846 within **%{** and **%}**), and a **typedef** used to define the symbol **YYSTYPE** to represent this union.
 116847 The effect of **%union** is to provide the declaration of **YYSTYPE** directly from the *yacc* input.

116848 C-language declarations and definitions can appear in the declarations section, enclosed by the
 116849 following marks:

```
116850 %{ ... %}
```

116851 These statements shall be copied into the code file, and have global scope within it so that they
 116852 can be used in the rules and program sections. The statements shall not contain **"%}"** outside a
 116853 comment, string literal, or multi-character constant.

116854 The application shall ensure that the declarations section is terminated by the token `%%`.

116855 Grammar Rules in yacc

116856 The rules section defines the context-free grammar to be accepted by the function `yacc` generates,
116857 and associates with those rules C-language actions and additional precedence information. The
116858 grammar is described below, and a formal definition follows.

116859 The rules section is comprised of one or more grammar rules. A grammar rule has the form:

```
116860 A : BODY ;
```

116861 The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or
116862 more *names*, *literals*, and *semantic actions* that can then be followed by optional *precedence rules*.
116863 Only the names and literals participate in the formation of the grammar; the semantic actions
116864 and precedence rules are used in other ways. The `<colon>` and the `<semicolon>` are `yacc`
116865 punctuation. If there are several successive grammar rules with the same left-hand side, the
116866 `<vertical-line>` (`'|'`) can be used to avoid rewriting the left-hand side; in this case the
116867 `<semicolon>` appears only after the last rule. The **BODY** part can be empty (or empty of names
116868 and literals) to indicate that the non-terminal symbol matches the empty string.

116869 The `yacc` utility assigns a unique number to each rule. Rules using the vertical bar notation are
116870 distinct rules. The number assigned to the rule appears in the description file.

116871 The elements comprising a **BODY** are:

116872 *name, literal* These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*
116873 stands for itself (less the lexically required quotation marks).

116874 *semantic action*

116875 With each grammar rule, the user can associate actions to be performed each time
116876 the rule is recognized in the input process. (Note that the word "action" can also
116877 refer to the actions of the parser—shift, reduce, and so on.)

116878 These actions can return values and can obtain the values returned by previous
116879 actions. These values are kept in objects of type `YYSTYPE` (see `%union`). The
116880 result value of the action shall be kept on the parse stack with the left-hand side of
116881 the rule, to be accessed by other reductions as part of their right-hand side. By
116882 using the `<tag>` information provided in the declarations section, the code
116883 generated by `yacc` can be strictly type checked and contain arbitrary information. In
116884 addition, the lexical analyzer can provide the same kinds of values for tokens, if
116885 desired.

116886 An action is an arbitrary C statement and as such can do input or output, call
116887 subprograms, and alter external variables. An action is one or more C statements
116888 enclosed in curly braces `'{'` and `'}'`. The statements shall not contain
116889 unbalanced curly brace preprocessing tokens.

116890 Certain pseudo-variables can be used in the action. These are macros for access to
116891 data structures known internally to `yacc`.

116892 `$$` The value of the action can be set by assigning it to `$$`. If type
116893 checking is enabled and the type of the value to be assigned cannot
116894 be determined, a diagnostic message may be generated.

116895 `$number` This refers to the value returned by the component specified by the
116896 token *number* in the right side of a rule, reading from left to right;
116897 *number* can be zero or negative. If *number* is zero or negative, it refers

116898 to the data associated with the name on the parser's stack preceding
 116899 the leftmost symbol of the current rule. (That is, "\$0" refers to the
 116900 name immediately preceding the leftmost name in the current rule to
 116901 be found on the parser's stack and "\$-1" refers to the symbol to *its*
 116902 left.) If *number* refers to an element past the current point in the rule,
 116903 or beyond the bottom of the stack, the result is undefined. If type
 116904 checking is enabled and the type of the value to be assigned cannot
 116905 be determined, a diagnostic message may be generated.

116906 \$<tag>number

116907 These correspond exactly to the corresponding symbols without the
 116908 *tag* inclusion, but allow for strict type checking (and preclude
 116909 unwanted type conversions). The effect is that the macro is expanded
 116910 to use *tag* to select an element from the YYSTYPE union (using
 116911 *dataname.tag*). This is particularly useful if *number* is not positive.

116912 \$<tag>\$

116913 This imposes on the reference the type of the union member
 116914 referenced by *tag*. This construction is applicable when a reference to
 116915 a left context value occurs in the grammar, and provides *yacc* with a
 means for selecting a type.

116916 Actions can occur anywhere in a rule (not just at the end); an action can access
 116917 values returned by actions to its left, and in turn the value it returns can be
 116918 accessed by actions to its right. An action appearing in the middle of a rule shall be
 116919 equivalent to replacing the action with a new non-terminal symbol and adding an
 116920 empty rule with that non-terminal symbol on the left-hand side. The semantic
 116921 action associated with the new rule shall be equivalent to the original action. The
 116922 use of actions within rules might introduce conflicts that would not otherwise
 116923 exist.

116924 By default, the value of a rule shall be the value of the first element in it. If the first
 116925 element does not have a type (particularly in the case of a literal) and type
 116926 checking is turned on by **%type**, an error message shall result.

116927 *precedence* The keyword **%prec** can be used to change the precedence level associated with a
 116928 particular grammar rule. Examples of this are in cases where a unary and binary
 116929 operator have the same symbolic representation, but need to be given different
 116930 precedences, or where the handling of an ambiguous if-else construction is
 116931 necessary. The reserved symbol **%prec** can appear immediately after the body of
 116932 the grammar rule and can be followed by a token name or a literal. It shall cause
 116933 the precedence of the grammar rule to become that of the following token name or
 116934 literal. The action for the rule as a whole can follow **%prec**.

116935 If a program section follows, the application shall ensure that the grammar rules are terminated
 116936 by **%%**.

116937 **Programs Section**

116938 The *programs* section can include the definition of the lexical analyzer *yylex()*, and any other
 116939 functions; for example, those used in the actions specified in the grammar rules. It is unspecified
 116940 whether the programs section precedes or follows the semantic actions in the output file;
 116941 therefore, if the application contains any macro definitions and declarations intended to apply to
 116942 the code in the semantic actions, it shall place them within "%{ . . . %}" in the declarations
 116943 section.

116944 **Input Grammar**

116945 The following input to *yacc* yields a parser for the input to *yacc*. This formal syntax takes
 116946 precedence over the preceding text syntax description.

116947 The lexical structure is defined less precisely; [Lexical Structure of the Grammar](#) (on page 3457)
 116948 defines most terms. The correspondence between the previous terms and the tokens below is as
 116949 follows.

116950 **IDENTIFIER** This corresponds to the concept of *name*, given previously. It also includes
 116951 literals as defined previously.

116952 **C_IDENTIFIER** This is a name, and additionally it is known to be followed by a <colon>. A
 116953 literal cannot yield this token.

116954 **NUMBER** A string of digits (a non-negative decimal integer).

116955 **TYPE, LEFT, MARK, LCURL, RCURL**

116956 These correspond directly to **%type**, **%left**, **%%**, **%{**, and **%}**.

116957 **{...}** This indicates C-language source code, with the possible inclusion of '\$'
 116958 macros as discussed previously.

```

116959 /* Grammar for the input to yacc. */
116960 /* Basic entries. */
116961 /* The following are recognized by the lexical analyzer. */
116962 %token IDENTIFIER /* Includes identifiers and literals */
116963 %token C_IDENTIFIER /* identifier (but not literal)
116964 followed by a :. */
116965 %token NUMBER /* [0-9][0-9]* */
116966 /* Reserved words : %type=>TYPE %left=>LEFT, and so on */
116967 %token LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION
116968 %token MARK /* The %% mark. */
116969 %token LCURL /* The %{ mark. */
116970 %token RCURL /* The %} mark. */
116971 /* 8-bit character literals stand for themselves; */
116972 /* tokens have to be defined for multi-byte characters. */
116973 %start spec
116974 %%
116975 spec : defs MARK rules tail
116976 ;
116977 tail : MARK
116978 {
116979 /* In this action, set up the rest of the file. */
116980 }
116981 | /* Empty; the second MARK is optional. */
116982 ;
116983 defs : /* Empty. */
116984 | defs def
116985 ;
116986 def : START IDENTIFIER
116987 | UNION

```

```

116988     {
116989         /* Copy union definition to output. */
116990     }
116991     |    LCURL
116992     {
116993         /* Copy C code to output file. */
116994     }
116995     RCURL
116996     |    rword tag nlist
116997     ;
116998 rword : TOKEN
116999     | LEFT
117000     | RIGHT
117001     | NONASSOC
117002     | TYPE
117003     ;
117004 tag  : /* Empty: union tag ID optional. */
117005     | '<' IDENTIFIER '>'
117006     ;
117007 nlist : nmno
117008     | nlist nmno
117009     ;
117010 nmno  : IDENTIFIER          /* Note: literal invalid with % type. */
117011     | IDENTIFIER NUMBER    /* Note: invalid with % type. */
117012     ;
117013 /* Rule section */
117014 rules : C_IDENTIFIER rbody prec
117015     | rules rule
117016     ;
117017 rule  : C_IDENTIFIER rbody prec
117018     | '|' rbody prec
117019     ;
117020 rbody : /* empty */
117021     | rbody IDENTIFIER
117022     | rbody act
117023     ;
117024 act   : '{'
117025         {
117026             /* Copy action, translate $$, and so on. */
117027         }
117028         '}'
117029     ;
117030 prec  : /* Empty */
117031     | PREC IDENTIFIER
117032     | PREC IDENTIFIER act
117033     | prec ';'
117034     ;

```

117035 Conflicts

117036 The parser produced for an input grammar may contain states in which conflicts occur. The
 117037 conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at
 117038 least one LALR(1) conflict. The *yacc* utility shall resolve all conflicts, using either default rules or
 117039 user-specified precedence rules.

117040 Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is
 117041 where, for a given state and lookahead symbol, both a shift action and a reduce action are
 117042 possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions
 117043 by two different rules are possible.

117044 The rules below describe how to specify what actions to take when a conflict occurs. Not all
 117045 shift/reduce conflicts can be successfully resolved this way because the conflict may be due to
 117046 something other than ambiguity, so incautious use of these facilities can cause the language
 117047 accepted by the parser to be much different from that which was intended. The description file
 117048 shall contain sufficient information to understand the cause of the conflict. Where ambiguity is
 117049 the reason either the default or explicit rules should be adequate to produce a working parser.

117050 The declared precedences and associativities (see [Declarations Section](#), on page 3457) are used to
 117051 resolve parsing conflicts as follows:

- 117052 1. A precedence and associativity is associated with each grammar rule; it is the precedence
 117053 and associativity of the last token or literal in the body of the rule. If the **%prec** keyword
 117054 is used, it overrides this default. Some grammar rules might not have both precedence
 117055 and associativity.
- 117056 2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have
 117057 precedence and associativity associated with them, then the conflict is resolved in favor of
 117058 the action (shift or reduce) associated with the higher precedence. If the precedences are
 117059 the same, then the associativity is used; left associative implies reduce, right associative
 117060 implies shift, and non-associative implies an error in the string being parsed.
- 117061 3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done.
 117062 Conflicts resolved this way are counted in the diagnostic output described in [Error
 117063 Handling](#).
- 117064 4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that
 117065 occurs earlier in the input sequence. Conflicts resolved this way are counted in the
 117066 diagnostic output described in [Error Handling](#).

117067 Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and
 117068 reduce/reduce conflicts reported by *yacc* on either standard error or in the description file.

117069 Error Handling

117070 The token **error** shall be reserved for error handling. The name **error** can be used in grammar
 117071 rules. It indicates places where the parser can recover from a syntax error. The default value of
 117072 **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer
 117073 should not return the value of **error**.

117074 The parser shall detect a syntax error when it is in a state where the action associated with the
 117075 lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by
 117076 executing the macro YYERROR. When YYERROR is executed, the semantic action passes control
 117077 back to the parser. YYERROR cannot be used outside of semantic actions.

117078 When the parser detects a syntax error, it normally calls *yyerror()* with the character string
 117079 "syntax error" as its argument. The call shall not be made if the parser is still recovering

117080 from a previous error when the error is detected. The parser is considered to be recovering from
117081 a previous error until the parser has shifted over at least three normal input symbols since the
117082 last error was detected or a semantic action has executed the macro *yyerrok*. The parser shall not
117083 call *yyerror()* when YYERROR is executed.

117084 The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the
117085 parser has not yet fully recovered from it. Otherwise, zero shall be returned.

117086 When a syntax error is detected by the parser, the parser shall check if a previous syntax error
117087 has been detected. If a previous error was detected, and if no normal input symbols have been
117088 shifted since the preceding error was detected, the parser checks if the lookahead symbol is an
117089 endmarker (see [Interface to the Lexical Analyzer](#)). If it is, the parser shall return with a non-zero
117090 value. Otherwise, the lookahead symbol shall be discarded and normal parsing shall resume.

117091 When YYERROR is executed or when the parser detects a syntax error and no previous error has
117092 been detected, or at least one normal input symbol has been shifted since the previous error was
117093 detected, the parser shall pop back one state at a time until the parse stack is empty or the
117094 current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a
117095 non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser
117096 reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead
117097 symbol when parsing is resumed.

117098 The macro *yyerrok* in a semantic action shall cause the parser to act as if it has fully recovered
117099 from any previous errors. The macro *yyclearin* shall cause the parser to discard the current
117100 lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no
117101 effect.

117102 The macro YYACCEPT shall cause the parser to return with the value zero. The macro
117103 YYABORT shall cause the parser to return with a non-zero value.

117104 **Interface to the Lexical Analyzer**

117105 The *yylex()* function is an integer-valued function that returns a *token number* representing the
117106 kind of token read. If there is a value associated with the token returned by *yylex()* (see the
117107 discussion of *tag* above), it shall be assigned to the external variable *yyval*.

117108 If the parser and *yylex()* do not agree on these token numbers, reliable communication between
117109 them cannot occur. For (single-byte character) literals, the token is simply the numeric value of
117110 the character in the current character set. The numbers for other tokens can either be chosen by
117111 *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex()*
117112 to return these numbers symbolically. The **#define** statements are put into the code file, and the
117113 header file if that file is requested. The set of characters permitted by *yacc* in an identifier is
117114 larger than that permitted by C. Token names found to contain such characters shall not be
117115 included in the **#define** declarations.

117116 If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers
117117 greater than 256, although no order is implied. A token can be explicitly assigned a number by
117118 following its first appearance in the declarations section with a number. Names and literals not
117119 defined this way retain their default definition. All token numbers assigned by *yacc* shall be
117120 unique and distinct from the token numbers used for literals and user-assigned tokens. If
117121 duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error;
117122 otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

117123 The end of the input is marked by a special token called the *endmarker*, which has a token
117124 number that is zero or negative. (These values are invalid for any other token.) All lexical
117125 analyzers shall return zero or negative as a token number upon reaching the end of their input.

117126 If the tokens up to, but excluding, the endmarker form a structure that matches the start symbol,
117127 the parser shall accept the input. If the endmarker is seen in any other context, it shall be
117128 considered an error.

117129 **Completing the Program**

117130 In addition to *yyparse()* and *yylex()*, the functions *yyerror()* and *main()* are required to make a
117131 complete program. The application can supply *main()* and *yyerror()*, or those routines can be
117132 obtained from the *yacc* library.

117133 **Yacc Library**

117134 The following functions shall appear only in the *yacc* library accessible through the `-I y` operand
117135 to *c99*; they can therefore be redefined by a conforming application:

117136 **int main(void)**

117137 This function shall call *yyparse()* and exit with an unspecified value. Other actions within
117138 this function are unspecified.

117139 **int yyerror(const char *s)**

117140 This function shall write the NUL-terminated argument to standard error, followed by a
117141 <newline>.

117142 The order of the `-I y` and `-I l` operands given to *c99* is significant; the application shall either
117143 provide its own *main()* function or ensure that `-I y` precedes `-I l`.

117144 **Debugging the Parser**

117145 The parser generated by *yacc* shall have diagnostic facilities in it that can be optionally enabled
117146 at either compile time or at runtime (if enabled at compile time). The compilation of the runtime
117147 debugging code is under the control of *YYDEBUG*, a preprocessor symbol. If *YYDEBUG* has a
117148 non-zero value, the debugging code shall be included. If its value is zero, the code shall not be
117149 included.

117150 In parsers where the debugging code has been included, the external **int** *yydebug* can be used to
117151 turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of
117152 *yydebug* shall be zero.

117153 When `-t` is specified, the code file shall be built such that, if *YYDEBUG* is not already defined at
117154 compilation time (using the *c99* `-D YYDEBUG` option, for example), *YYDEBUG* shall be set
117155 explicitly to 1. When `-t` is not specified, the code file shall be built such that, if *YYDEBUG* is not
117156 already defined, it shall be set explicitly to zero.

117157 The format of the debugging output is unspecified but includes at least enough information to
117158 determine the shift and reduce actions, and the input symbols. It also provides information
117159 about error recovery.

117160 **Algorithms**

117161 The parser constructed by *yacc* implements an LALR(1) parsing algorithm as documented in the
117162 literature. It is unspecified whether the parser is table-driven or direct-coded.

117163 A parser generated by *yacc* shall never request an input symbol from *yylex()* while in a state
117164 where the only actions other than the error action are reductions by a single rule.

117165 The literature of parsing theory defines these concepts.

117166 **Limits**

117167 The *yacc* utility may have several internal tables. The minimum maximums for these tables are
 117168 shown in the following table. The exact meaning of these values is implementation-defined. The
 117169 implementation shall define the relationship between these values and between them and any
 117170 error messages that the implementation may generate should it run out of space for any internal
 117171 structure. An implementation may combine groups of these resources into a single pool as long
 117172 as the total available to the user does not fall below the sum of the sizes specified by this section.

117173 **Table 4-23** Internal Limits in *yacc*

Limit	Minimum Maximum	Description
{NTERMS}	126	Number of tokens.
{NNONTERM}	200	Number of non-terminals.
{NPROD}	300	Number of rules.
{NSTATES}	600	Number of states.
{MEMSIZE}	5 200	Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in Grammar Rules in yacc (on page 3459).
{ACTSIZE}	4 000	Number of actions. ``Actions'' here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in Grammar Rules in yacc (on page 3459).

117191 **EXIT STATUS**

117192 The following exit values shall be returned:

117193 0 Successful completion.

117194 >0 An error occurred.

117195 **CONSEQUENCES OF ERRORS**

117196 If any errors are encountered, the run is aborted and *yacc* exits with a non-zero status. Partial
 117197 code files and header files may be produced. The summary information in the description file
 117198 shall always be produced if the `-v` flag is present.

117199 **APPLICATION USAGE**

117200 Historical implementations experience name conflicts on the names `yacc.tmp`, `yacc.acts`,
 117201 `yacc.debug`, `y.tab.c`, `y.tab.h`, and `y.output` if more than one copy of *yacc* is running in a single
 117202 directory at one time. The `-b` option was added to overcome this problem. The related problem
 117203 of allowing multiple *yacc* parsers to be placed in the same file was addressed by adding a `-p`
 117204 option to override the previously hard-coded `yy` variable prefix.

117205 The description of the `-p` option specifies the minimal set of function and variable names that
 117206 cause conflict when multiple parsers are linked together. `YYSTYPE` does not need to be changed.
 117207 Instead, the programmer can use `-b` to give the header files for different parsers different names,
 117208 and then the file with the `yylex()` for a given parser can include the header for that parser.
 117209 Names such as `yyclearerr` do not need to be changed because they are used only in the actions;
 117210 they do not have linkage. It is possible that an implementation has other names, either internal
 117211 ones for implementing things such as `yyclearerr`, or providing non-standard features that it wants

117212 to change with `-p`.

117213 Unary operators that are the same token as a binary operator in general need their precedence
117214 adjusted. This is handled by the `%prec` advisory symbol associated with the particular grammar
117215 rule defining that unary operator. (See [Grammar Rules in yacc](#) (on page 3459).) Applications are
117216 not required to use this operator for unary operators, but the grammars that do not require it are
117217 rare.

117218 EXAMPLES

117219 Access to the *yacc* library is obtained with library search operands to *c99*. To use the *yacc* library
117220 *main()*:

```
117221 c99 y.tab.c -l y
```

117222 Both the *lex* library and the *yacc* library contain *main()*. To access the *yacc main()*:

```
117223 c99 y.tab.c lex.yy.c -l y -l l
```

117224 This ensures that the *yacc* library is searched first, so that its *main()* is used.

117225 The historical *yacc* libraries have contained two simple functions that are normally coded by the
117226 application programmer. These functions are similar to the following code:

```
117227 #include <locale.h>
117228 int main(void)
117229 {
117230     extern int yyparse();
117231     setlocale(LC_ALL, "");
117232     /* If the following parser is one created by lex, the
117233        application must be careful to ensure that LC_CTYPE
117234        and LC_COLLATE are set to the POSIX locale. */
117235     (void) yyparse();
117236     return (0);
117237 }
117238 #include <stdio.h>
117239 int yyerror(const char *msg)
117240 {
117241     (void) fprintf(stderr, "%s\n", msg);
117242     return (0);
117243 }
```

117244 RATIONALE

117245 The references in **Referenced Documents** may be helpful in constructing the parser generator.
117246 The referenced DeRemer and Pennello article (along with the works it references) describes a
117247 technique to generate parsers that conform to this volume of POSIX.1-2017. Work in this area
117248 continues to be done, so implementors should consult current literature before doing any new
117249 implementations. The original Knuth article is the theoretical basis for this kind of parser, but the
117250 tables it generates are impractically large for reasonable grammars and should not be used. The
117251 “equivalent to” wording is intentional to assure that the best tables that are LALR(1) can be
117252 generated.

117253 There has been confusion between the class of grammars, the algorithms needed to generate
117254 parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal.
117255 In particular, a parser generator that accepts the full range of LR(1) grammars need not generate
117256 a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars)

117257 for a grammar that happens to be SLR(1). Such an implementation need not recognize the case,
117258 either; table compression can yield the SLR(1) table (or one even smaller than that) without
117259 recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent
117260 more upon the table representation and compression (or the code generation if a direct parser is
117261 generated) than upon the class of grammar that the table generator handles.

117262 The speed of the parser generator is somewhat dependent upon the class of grammar it handles.
117263 However, the original Knuth article algorithms for constructing LR parsers were judged by its
117264 author to be impractically slow at that time. Although full LR is more complex than LALR(1), as
117265 computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock
117266 execution time) is becoming less significant.

117267 Potential authors are cautioned that the referenced DeRemer and Pennello article previously
117268 cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in
117269 some of the LALR(1) algorithm statements that preceded it to publication. They should take the
117270 time to seek out that paper, as well as current relevant work, particularly Aho's.

117271 The **-b** option was added to provide a portable method for permitting *yacc* to work on multiple
117272 separate parsers in the same directory. If a directory contains more than one *yacc* grammar, and
117273 both grammars are constructed at the same time (by, for example, a parallel *make* program),
117274 conflict results. While the solution is not historical practice, it corrects a known deficiency in
117275 historical implementations. Corresponding changes were made to all sections that referenced
117276 the filenames **y.tab.c** (now ``the code file''), **y.tab.h** (now ``the header file''), and **y.output** (now
117277 ``the description file").

117278 The grammar for *yacc* input is based on System V documentation. The textual description shows
117279 there that the **' ; '** is required at the end of the rule. The grammar and the implementation do
117280 not require this. (The use of **C_IDENTIFIER** causes a reduce to occur in the right place.)

117281 Also, in that implementation, the constructs such as **%token** can be terminated by a
117282 <semicolon>, but this is not permitted by the grammar. The keywords such as **%token** can also
117283 appear in uppercase, which is again not discussed. In most places where **'%'** is used,
117284 <backslash> can be substituted, and there are alternate spellings for some of the symbols (for
117285 example, **%LEFT** can be **"%<"** or even **"\<"**).

117286 Historically, <tag> can contain any characters except **'>'**, including white space, in the
117287 implementation. However, since the *tag* must reference an ISO C standard union member, in
117288 practice conforming implementations need to support only the set of characters for ISO C
117289 standard identifiers in this context.

117290 Some historical implementations are known to accept actions that are terminated by a period.
117291 Historical implementations often allow **'\$'** in names. A conforming implementation does not
117292 need to support either of these behaviors.

117293 Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of *yacc*. There
117294 may be situations in which the *grammar* is not, strictly speaking, in error, and yet *yacc* cannot
117295 interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances
117296 be resolved by providing additional information, such as using **%type** or **%union** declarations.
117297 It is often easier and it usually yields a smaller parser to take this alternative when it is
117298 appropriate.

117299 The size and execution time of a program produced without the runtime debugging code is
117300 usually smaller and slightly faster in historical implementations.

117301 Statistics messages from several historical implementations include the following types of
117302 information:

117303 *n*/512 terminals, *n*/300 non-terminals
 117304 *n*/600 grammar rules, *n*/1500 states
 117305 *n* shift/reduce, *n* reduce/reduce conflicts reported
 117306 *n*/350 working sets used
 117307 Memory: states, etc. *n*/15 000, parser *n*/15 000
 117308 *n*/600 distinct lookahead sets
 117309 *n* extra closures
 117310 *n* shift entries, *n* exceptions
 117311 *n* goto entries
 117312 *n* entries saved by goto default
 117313 Optimizer space used: input *n*/15 000, output *n*/15 000
 117314 *n* table entries, *n* zero
 117315 Maximum spread: *n*, Maximum offset: *n*

117316 The report of internal tables in the description file is left implementation-defined because all
 117317 aspects of these limits are also implementation-defined. Some implementations may use
 117318 dynamic allocation techniques and have no specific limit values to report.

117319 The format of the **y.output** file is not given because specification of the format was not seen to
 117320 enhance applications portability. The listing is primarily intended to help human users
 117321 understand and debug the parser; use of **y.output** by a conforming application script would be
 117322 unusual. Furthermore, implementations have not produced consistent output and no popular
 117323 format was apparent. The format selected by the implementation should be human-readable, in
 117324 addition to the requirement that it be a text file.

117325 Standard error reports are not specifically described because they are seldom of use to
 117326 conforming applications and there was no reason to restrict implementations.

117327 Some implementations recognize "`={"`" as equivalent to "`'{'`" because it appears in historical
 117328 documentation. This construction was recognized and documented as obsolete as long ago as
 117329 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of POSIX.1-2017 chose to
 117330 leave it as obsolete and omit it.

117331 Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They
 117332 should not be returned as multi-byte character literals. The token **error** that is used for error
 117333 recovery is normally assigned the value 256 in the historical implementation. Thus, the token
 117334 value 256, which is used in many multi-byte character sets, is not available for use as the value
 117335 of a user-defined token.

117336 **FUTURE DIRECTIONS**

117337 None.

117338 **SEE ALSO**

117339 [c99](#), [lex](#)

117340 XBD [Chapter 8](#) (on page 173), [Section 12.2](#) (on page 216)

117341 **CHANGE HISTORY**

117342 First released in Issue 2.

117343 **Issue 5**

117344 The FUTURE DIRECTIONS section is added.

117345 **Issue 6**

117346 This utility is marked as part of the C-Language Development Utilities option.

117347 Minor changes have been added to align with the IEEE P1003.2b draft standard.

- 117348 The normative text is reworded to avoid use of the term “must” for application requirements.
- 117349 IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the }%
117350 token to the %}.
- 117351 **Issue 7**
- 117352 Austin Group Interpretation 1003.1-2001 #190 is applied, clarifying the requirements for
117353 generated code to conform to the ISO C standard.
- 117354 Austin Group Interpretation 1003.1-2001 #191 is applied, clarifying the handling of C-language
117355 trigraphs and curly brace preprocessing tokens.
- 117356 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
117357 apply.
- 117358 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 117359 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0204 [977] is applied.

117360 **NAME**

117361 zcat ‡expand and concatenate data

117362 **SYNOPSIS**117363 XSI zcat [*file...*]117364 **DESCRIPTION**

117365 The *zcat* utility shall write to standard output the uncompressed form of files that have been
 117366 compressed using the *compress* utility. It is the equivalent of *uncompress -c*. Input files are not
 117367 affected.

117368 **OPTIONS**

117369 None.

117370 **OPERANDS**

117371 The following operand shall be supported:

117372 *file* The pathname of a file previously processed by the *compress* utility. If *file* already
 117373 has the *.Z* suffix specified, it is used as submitted. Otherwise, the *.Z* suffix is
 117374 appended to the filename prior to processing.

117375 **STDIN**117376 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '- '.117377 **INPUT FILES**117378 Input files shall be compressed files that are in the format produced by the *compress* utility.117379 **ENVIRONMENT VARIABLES**117380 The following environment variables shall affect the execution of *zcat*:

117381 *LANG* Provide a default value for the internationalization variables that are unset or null.
 117382 (See XBD [Section 8.2](#) (on page 174) for the precedence of internationalization
 117383 variables used to determine the values of locale categories.)

117384 *LC_ALL* If set to a non-empty string value, override the values of all the other
 117385 internationalization variables.

117386 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 117387 characters (for example, single-byte as opposed to multi-byte characters in
 117388 arguments).

117389 *LC_MESSAGES*

117390 Determine the locale that should be used to affect the format and contents of
 117391 diagnostic messages written to standard error.

117392 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

117393 **ASYNCHRONOUS EVENTS**

117394 Default.

117395 **STDOUT**

117396 The compressed files given as input shall be written on standard output in their uncompressed
 117397 form.

117398 **STDERR**

117399 The standard error shall be used only for diagnostic messages.

117400 **OUTPUT FILES**

117401 None.

117402 **EXTENDED DESCRIPTION**

117403 None.

117404 **EXIT STATUS**

117405 The following exit values shall be returned:

117406 0 Successful completion.

117407 >0 An error occurred.

117408 **CONSEQUENCES OF ERRORS**

117409 Default.

117410 **APPLICATION USAGE**

117411 None.

117412 **EXAMPLES**

117413 None.

117414 **RATIONALE**

117415 None.

117416 **FUTURE DIRECTIONS**

117417 None.

117418 **SEE ALSO**117419 *compress, uncompress*117420 XBD [Chapter 8](#) (on page 173)117421 **CHANGE HISTORY**

117422 First released in Issue 4.

117423

 *Open Group Standard*

117424

Vol. 4:

117425

Rationale (Informative), Issue 7


117426

The Open Group

117427

The Institute of Electrical and Electronics Engineers, Inc.

117428

 *Rationale (Informative)*

117429

Part A:

117430

Base Definitions

117431

The Open Group

117432

The Institute of Electrical and Electronics Engineers, Inc.

Rationale for Base Definitions

117435 A.1 Introduction

117436 A.1.1 Scope

117437 POSIX.1-2017 is one of a family of standards known as POSIX. The family of standards extends
117438 to many topics; POSIX.1 consists of both operating system interfaces and shell and utilities.
117439 POSIX.1-2017 is technically identical to The Open Group Base Specifications, Issue 7.

117440 Scope of POSIX.1-2017

117441 The (paraphrased) goals of this development were to revise the single document that is ISO/IEC
117442 9945:2003 Parts 1 through 4, IEEE Std 1003.1, 2004 Edition, and the appropriate parts of The
117443 Open Group Single UNIX Specification, Version 3. This work has been undertaken by the
117444 Austin Group, a joint working group of IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

117445 The following are the base documents in this version:

117446 IEEE Std 1003.1, 2004 Edition

117447 ISO/IEC 9899:1999, Programming Language C, including ISO/IEC
117448 9899:1999/Cor.1:2001(E), ISO/IEC 9899:1999/Cor.2:2004(E), and ISO/IEC
117449 9899:1999/Cor.3

117450 The Open Group Extended API Sets, Parts 1 through 4

117451 This version has addressed the following areas:

117452 Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1,
117453 and ISO/IEC defect reports against ISO/IEC 9945

117454 The repository of interpretations can be accessed at www.opengroup.org/austin/interps.

117455 Issues raised in corrigenda for The Open Group Technical Standards and working group
117456 resolutions from The Open Group

117457 Issues arising from ISO TR 24715:2006, Conflicts between POSIX and the LSB

117458 This is a Type 3 informative technical report highlighting differences between the LSB 3.1
117459 and the 2004 Edition of this standard.

117460 Changes to make the text self-consistent with the additional material merged

117461 The new material merged has come from the The Open Group Extended API Sets, Parts 1
117462 through 4. A list of the new interfaces is included in [Section B.1.1](#) (on page 3561).

117463 Features, marked legacy or obsolescent in the base documents, have been considered for
117464 removal in this version

117465 See [Section B.1.1](#) (on page 3561) and [Section C.1.1](#) (on page 3707).

- 117466 A review and reorganization of the options within the standard
- 117467 This has included marking the following options obsolescent:
- 117468 † ~~at~~ Environment Services and Utilities
- 117469 † ~~at~~ing
- 117470 † ~~SIX~~STREAMS
- 117471 The UUCP Utilities option is a new option for this version.
- 117472 Functionality from the following former options is now mandatory in this version:
- 117473 AIO `_POSIX_ASYNCHRONOUS_IO` (Asynchronous Input and Output)
- 117474 BAR `_POSIX_BARRIERS` (Barriers)
- 117475 CS `_POSIX_CLOCK_SELECTION` (Clock Selection)
- 117476 MF `_POSIX_MAPPED_FILES` (Memory Mapped Files)
- 117477 MPR `_POSIX_MEMORY_PROTECTION` (Memory Protection)
- 117478 RTS `_POSIX_REALTIME_SIGNALS` (Realtime Signals Extension)
- 117479 RWL `_POSIX_READER_WRITER_LOCKS` (Read-Write Locks)
- 117480 SEM `_POSIX_SEMAPHORES` (Semaphores)
- 117481 SPI `_POSIX_SPIN_LOCKS` (Spin Locks)
- 117482 THR `_POSIX_THREADS` (Threads)
- 117483 TMO `_POSIX_TIMEOUTS` (Timeouts)
- 117484 TMR `_POSIX_TIMERS` (Timers)
- 117485 TSF `_POSIX_THREAD_SAFE_FUNCTIONS` (Thread-Safe Functions)
- 117486 Alignment with the ISO/IEC 9899:1999 standard, including ISO/IEC
- 117487 9899:1999/Cor.2:2004(E)
- 117488 A review of the use of fixed path filenames within the standard
- 117489 For example, the *at*, *batch*, and *crontab* utilities previously had a requirement for use of the
- 117490 directory **/usr/lib/cron**.
- 117491 The following were requirements on POSIX.1-2017:
- 117492 Backward-compatibility
- 117493 For interfaces carried forward, it was agreed that there should be no breakage of
- 117494 functionality in the existing base documents. All strictly conforming applications will be
- 117495 conforming but not necessarily strictly conforming to the revised standard. The goal is for
- 117496 system implementations to be able to support the existing and revised standards
- 117497 simultaneously.
- 117498 Architecture and *n*-bit-neutral
- 117499 The common standard should not make any implicit assumptions about the system
- 117500 architecture or size of data types; for example, previously some 32-bit implicit assumptions
- 117501 had crept into the standards.
- 117502 Extensibility
- 117503 It should be possible to extend the common standard without breaking backwards-
- 117504 compatibility; for example, the name space should be reserved and structured to avoid
- 117505 duplication of names between the standard and extensions to it.

117506 POSIX.1 and the ISO C Standard

117507 The standard developers believed it essential for a programmer to have a single complete
117508 reference place, but recognized that deference to the formal standard has to be addressed for the
117509 duplicate interface definitions between the ISO C standard and POSIX.1-2017.

117510 Where an interface has a version in the ISO C standard, the DESCRIPTION section describes the
117511 relationship to the ISO C standard and markings are included as appropriate to show where the
117512 ISO C standard has been extended in the text.

117513 A block of text is included at the start of each affected reference page stating whether the page is
117514 aligned with the ISO C standard or extended. Each page has been parsed for additions beyond
117515 the ISO C standard (that is, including both POSIX and UNIX extensions), and these extensions
117516 are marked as CX extensions (for C extensions).

117517 FIPS Requirements

117518 The Federal Information Processing Standards (FIPS) are a series of US government
117519 procurement standards managed and maintained on behalf of the US Department of Commerce
117520 by the National Institute of Standards and Technology (NIST).

117521 The following restrictions were integrated into IEEE Std 1003.1-2001. They originally came from
117522 FIPS 151-2 which was withdrawn by NIST on February 25 2000.

117523 The implementation supports `_POSIX_CHOWN_RESTRICTED`.

117524 The limit `{NGROUPS_MAX}` is greater than or equal to 8.

117525 The implementation supports the setting of the group ID of a file (when it is created) to
117526 that of the parent directory.

117527 The implementation supports `_POSIX_SAVED_IDS`.

117528 The implementation supports `_POSIX_VDISABLE`.

117529 The implementation supports `_POSIX_JOB_CONTROL`.

117530 The implementation supports `_POSIX_NO_TRUNC`.

117531 The `read()` function returns the number of bytes read when interrupted by a signal and
117532 does not return `-1`.

117533 The `write()` function returns the number of bytes written when interrupted by a signal and
117534 does not return `-1`.

117535 In the environment for the login shell, the environment variables `LOGNAME` and `HOME`
117536 are defined and have the properties described in POSIX.1-2017.

117537 The value of `{CHILD_MAX}` is greater than or equal to 25.

117538 The value of `{OPEN_MAX}` is greater than or equal to 20.

117539 The implementation supports the functionality associated with the symbols `CS7`, `CS8`,
117540 `CSTOPB`, `PARODD`, and `PARENB` defined in `<termios.h>`.

117541 **A.1.2 Conformance**117542 See [Section A.2](#) (on page 3483).117543 **A.1.3 Normative References**

117544 There is no additional rationale provided for this section.

117545 **A.1.4 Change History**

117546 For Issue 7 onwards, in references to Technical Corrigenda, the original Austin Group defect
 117547 report numbers that gave rise to the change are included in square brackets after the change
 117548 number from the Technical Corrigendum. For more information on Austin Group defect reports
 117549 see www.opengroup.org/austin/defectform.html.

117550 **A.1.5 Terminology**

117551 The meanings specified in POSIX.1-2017 for the words *shall*, *should*, and *may* are mandated by
 117552 ISO/IEC directives.

117553 In the Rationale (Informative) volume of POSIX.1-2017, the words *shall*, *should*, and *may* are
 117554 sometimes used to illustrate similar usages in POSIX.1-2017. However, the rationale itself does
 117555 not specify anything regarding implementations or applications.

117556 **conformance document**

117557 As a practical matter, the conformance document is effectively part of the system
 117558 documentation. Conformance documents are distinguished by POSIX.1-2017 so that they
 117559 can be referred to distinctly.

117560 **implementation-defined**

117561 This definition is analogous to that of the ISO C standard and, together with “undefined”
 117562 and “unspecified”, provides a range of specification of freedom allowed to the interface
 117563 implementor.

117564 **may**

117565 The use of *may* has been limited as much as possible, due both to confusion stemming from
 117566 its ordinary English meaning and to objections regarding the desirability of having as few
 117567 options as possible and those as clearly specified as possible.

117568 The usage of *can* and *may* were selected to contrast optional application behavior (can)
 117569 against optional implementation behavior (may).

117570 **shall**

117571 Declarative sentences are sometimes used in POSIX.1-2017 as if they included the word
 117572 *shall*, and facilities thus specified are no less required. For example, the two statements:

- 117573 1. The *foo()* function shall return zero.
 117574 2. The *foo()* function returns zero.

117575 are meant to be exactly equivalent.

117576 **should**

117577 In POSIX.1-2017, the word *should* does not usually apply to the implementation, but rather
 117578 to the application. Thus, the important words regarding implementations are *shall*, which
 117579 indicates requirements, and *may*, which indicates options.

117580 **obsolescent**

117581 The term “obsolescent” means “do not use this feature in new applications”. A feature
 117582 noted as obsolescent is supported by all implementations, but may be removed in a future
 117583 version; new applications should not use these features. The obsolescence concept is not an
 117584 ideal solution, but was used as a method of increasing consensus: many more objections
 117585 would be heard from the user community if some of these historical features were suddenly
 117586 removed without the grace period obsolescence implies. The phrase “may be removed in a
 117587 future version” implies that the result of that consideration might in fact keep those features
 117588 indefinitely if the predominance of applications do not migrate away from them quickly.

117589 **legacy**

117590 The term “legacy” was included in earlier versions of this standard but is no longer used in
 117591 the current version.

117592 **system documentation**

117593 The system documentation should normally describe the whole of the implementation,
 117594 including any extensions provided by the implementation. Such documents normally
 117595 contain information at least as detailed as the specifications in POSIX.1-2017. Few
 117596 requirements are made on the system documentation, but the term is needed to avoid a
 117597 dangling pointer where the conformance document is permitted to point to the system
 117598 documentation.

117599 **undefined**

117600 See *implementation-defined*.

117601 **unspecified**

117602 See *implementation-defined*.

117603 The definitions for “unspecified” and “undefined” appear nearly identical at first
 117604 examination, but are not. The term “unspecified” means that a conforming application may
 117605 deal with the unspecified behavior, and it should not care what the outcome is. The term
 117606 “undefined” says that a conforming application should not do it because no definition is
 117607 provided for what it does (and implicitly it would care what the outcome was if it tried it).
 117608 It is important to remember, however, that if the syntax permits the statement at all, it must
 117609 have some outcome in a real implementation.

117610 Thus, the terms “undefined” and “unspecified” apply to the way the application should
 117611 think about the feature. In terms of the implementation, it is always “defined”—there is
 117612 always some result, even if it is an error. The implementation is free to choose the behavior
 117613 it prefers.

117614 This also implies that an implementation, or another standard, could specify or define the
 117615 result in a useful fashion. The terms apply to POSIX.1-2017 specifically.

117616 The term “implementation-defined” implies requirements for documentation that are not
 117617 required for “undefined” (or “unspecified”). Where there is no need for a conforming
 117618 program to know the definition, the term “undefined” is used, even though
 117619 “implementation-defined” could also have been used in this context. There could be a
 117620 fourth term, specifying “this standard does not say what this does; it is acceptable to define
 117621 it in an implementation, but it does not need to be documented”, and undefined would
 117622 then be used very rarely for the few things for which any definition is not useful. In
 117623 particular, implementation-defined is used where it is believed that certain classes of
 117624 application will need to know such details to determine whether the application can be
 117625 successfully ported to the implementation. Such applications are not always strictly
 117626 portable, but nevertheless are common and useful; often the requirements met by the
 117627 application cannot be met without dealing with the issues implied by “implementation-
 117628 defined”. In some places the text refers to facilities supplied by the implementation that are

117629 outside the standard as implementation-supplied or implementation-provided. This is not
 117630 intended to imply a requirement for documentation. If it were, the term “implementation-
 117631 defined” would have been used.

117632 In many places POSIX.1-2017 is silent about the behavior of some possible construct. For
 117633 example, a variable may be defined for a specified range of values and behaviors are
 117634 described for those values; nothing is said about what happens if the variable has any other
 117635 value. That kind of silence can imply an error in the standard, but it may also imply that the
 117636 standard was intentionally silent and that any behavior is permitted. There is a natural
 117637 tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent.
 117638 Silence is intended to be equivalent to the term “unspecified”.

117639 Three terms used within POSIX.1-2017 overlap in meaning: “macro”, “symbolic name”, and
 117640 “symbolic constant”.

117641 **macro**

117642 This usually describes a C preprocessor symbol, the result of the **#define** operator, with or
 117643 without an argument. It may also be used to describe similar mechanisms in editors and
 117644 text processors.

117645 **symbolic name**

117646 In earlier versions of this standard this was also sometimes used to refer to a C preprocessor
 117647 symbol (without arguments), but the intention is for all such uses to have been removed. It
 117648 is now mainly used to refer to the names for characters in character sets, but is sometimes
 117649 used to refer to host names and even filenames.

117650 **symbolic constant**

117651 This also refers to a C preprocessor symbol, with specific associated requirements. See the
 117652 definition in [Section 3.380](#) (on page 95).

117653 **A.1.6 Definitions and Concepts**

117654 There is no additional rationale provided for this section.

117655 **A.1.7 Portability**

117656 To aid the identification of options within POSIX.1-2017, a notation consisting of margin codes
 117657 and shading is used. This is based on the notation used in earlier versions of The Open Group
 117658 Base specifications.

117659 The benefit of this approach is a reduction in the number of *if* statements within the running
 117660 text, that makes the text easier to read, and also an identification to the programmer that they
 117661 need to ensure that their target platforms support the underlying options. For example, if
 117662 functionality is marked with RPP in the margin, it will be available on all systems supporting
 117663 the Robust Mutex Priority Protection option, but may not be available on some others.

117664 **A.1.7.1 Codes**

117665 This section includes codes for options defined in XBD [Section 2.1.6](#) (on page 26), and the
 117666 following additional codes for other purposes:

117667 **CX** This margin code is used to denote extensions beyond the ISO C standard. For
 117668 interfaces that are duplicated between POSIX.1-2017 and the ISO C standard, a CX
 117669 introduction block describes the nature of the duplication, with any extensions

- 117670 appropriately CX marked and shaded.
- 117671 Where an interface is added to an ISO C standard header, within the header the
117672 interface has an appropriate margin marker and shading (for example, CX, XSI, TSF,
117673 and so on) and the same marking appears on the reference page in the SYNOPSIS
117674 section. This enables a programmer to easily identify that the interface is extending an
117675 ISO C standard header.
- 117676 **MX and MXX**
- 117677 These two margin codes both relate to the IEC 60559 Floating-Point option. The MX
117678 code denotes functionality that is mandated by the ISO C standard for IEC 60559
117679 implementations; the MXX code denotes IEC 60559 functionality that is an extension to
117680 the ISO C standard.
- 117681 **OB** This margin code is used to denote obsolescent behavior and thus flag a possible future
117682 applications portability warning.
- 117683 **OH** The Single UNIX Specification has historically tried to reduce the number of headers an
117684 application has had to include when using a particular interface. Sometimes this was
117685 fewer than the base standard, and hence a notation is used to flag which headers are
117686 optional if you are using a system supporting the XSI option.
- 117687 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0001 [591] is applied.

117688 *A.1.7.2 Margin Code Notation*

- 117689 Since some features may depend on one or more options, or require more than one option, a
117690 notation is used. Where a feature requires support of a single option, a single margin code will
117691 occur in the margin. If it depends on two options and both are required, then the codes will
117692 appear with a <space> separator. If either of two options are required, then a logical OR is
117693 denoted using the ' | ' symbol. If more than two codes are used, a special notation is used.

117694 **A.2 Conformance**

- 117695 The terms “profile” and “profiling” are used throughout this section.
- 117696 A profile of a standard or standards is a codified set of option selections, such that by being
117697 conformant to a profile, particular classes of users are specifically supported.

117698 **A.2.1 Implementation Conformance**

- 117699 These definitions allow application developers to know what to depend on in an
117700 implementation.
- 117701 There is no definition of a “strictly conforming implementation”; that would be an
117702 implementation that provides *only* those facilities specified by POSIX.1 with no extensions
117703 whatsoever. This is because no actual operating system implementation can exist without
117704 system administration and initialization facilities that are beyond the scope of POSIX.1.

117705 A.2.1.1 *Requirements*

117706 The word “support” is used in certain instances, rather than “provide”, in order to allow an
117707 implementation that has no resident software development facilities, but that supports the
117708 execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

117709 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0002 [810] is applied.

117710 A.2.1.2 *Documentation*

117711 The conformance documentation is required to use the same numbering scheme as POSIX.1 for
117712 purposes of cross-referencing. All options that an implementation chooses are reflected in
117713 **<limits.h>** and **<unistd.h>**.

117714 Note that the use of “may” in terms of where conformance documents record where
117715 implementations may vary, implies that it is not required to describe those features identified as
117716 undefined or unspecified.

117717 Other aspects of systems must be evaluated by purchasers for suitability. Many systems
117718 incorporate buffering facilities, maintaining updated data in volatile storage and transferring
117719 such updates to non-volatile storage asynchronously. Various exception conditions, such as a
117720 power failure or a system crash, can cause this data to be lost. The data may be associated with a
117721 file that is still open, with one that has been closed, with a directory, or with any other internal
117722 system data structures associated with permanent storage. This data can be lost, in whole or
117723 part, so that only careful inspection of file contents could determine that an update did not
117724 occur.

117725 Also, interrelated file activities, where multiple files and/or directories are updated, or where
117726 space is allocated or released in the file system structures, can leave inconsistencies in the
117727 relationship between data in the various files and directories, or in the file system itself. Such
117728 inconsistencies can break applications that expect updates to occur in a specific sequence, so that
117729 updates in one place correspond with related updates in another place.

117730 For example, if a user creates a file, places information in the file, and then records this action in
117731 another file, a system or power failure at this point followed by restart may result in a state in
117732 which the record of the action is permanently recorded, but the file created (or some of its
117733 information) has been lost. The consequences of this to the user may be undesirable. For a user
117734 on such a system, the only safe action may be to require the system administrator to have a
117735 policy that requires, after any system or power failure, that the entire file system must be
117736 restored from the most recent backup copy (causing all intervening work to be lost).

117737 The characteristics of each implementation will vary in this respect and may or may not meet
117738 the requirements of a given application or user. Enforcement of such requirements is beyond the
117739 scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an
117740 implementation that affect the exposure to possible data or sequence loss, and also what
117741 underlying implementation techniques and/or facilities are provided that reduce or limit such
117742 loss or its consequences.

117743 A.2.1.3 *POSIX Conformance*

117744 This really means conformance to the base standard; however, since this document includes the
117745 core material of the Single UNIX Specification, the standard developers decided that it was
117746 appropriate to segment the conformance requirements into two, the former for the base
117747 standard, and the latter for the Single UNIX Specification (denoted XSI Conformance).

117748 Within POSIX.1 there are some symbolic constants that, if defined to a certain value or range of
117749 values, indicate that a certain option is enabled. Other symbolic constants exist in POSIX.1 for

117750 other reasons.

117751 In this version, some features that were previously optional have been made mandatory. For
117752 backwards compatibility, the symbolic constants associated with the option are still required
117753 now with fixed allowable ranges or values. The following options from the previous version of
117754 this standard are now mandatory:

117755 _POSIX_ASYNCHRONOUS_IO
117756 _POSIX_BARRIERS
117757 _POSIX_CLOCK_SELECTION
117758 _POSIX_MAPPED_FILES
117759 _POSIX_MEMORY_PROTECTION
117760 _POSIX_READER_WRITER_LOCKS
117761 _POSIX_REALTIME_SIGNALS
117762 _POSIX_SEMAPHORES
117763 _POSIX_SPIN_LOCKS
117764 _POSIX_THREAD_SAFE_FUNCTIONS
117765 _POSIX_THREADS
117766 _POSIX_TIMEOUTS
117767 _POSIX_TIMERS

117768 A POSIX-conformant system may support the XSI option required by the Single UNIX
117769 Specification. This was intentional since the standard developers intend them to be upwards-
117770 compatible, so that a system conforming to the Single UNIX Specification can also conform to
117771 the base standard at the same time.

117772 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0003 [637] is applied.

117773 A.2.1.4 XSI Conformance

117774 This section is included to describe the conformance requirements for the base volumes of the
117775 Single UNIX Specification.

117776 XSI conformance can be thought of as a profile, selecting certain options from POSIX.1-2017.

117777 A.2.1.5 Option Groups

117778 The concept of “Option Groups” is included to allow collections of related functions or options
117779 to be grouped together. This has been used as follows: the “XSI Option Groups” have been
117780 created to allow super-options, collections of underlying options and related functions, to be
117781 collectively supported by XSI-conforming systems.

117782 The standard developers considered the matter of subprofiling and decided it was better to
117783 include an enabling mechanism rather than detailed normative requirements. A set of
117784 subprofiling options was developed and included later in this volume of POSIX.1-2017 as an
117785 informative illustration.

117786 **Subprofiling Considerations**

117787 The goal of not simultaneously fixing maximums and minimums was to allow implementations
 117788 of the base standard or standards to support multiple profiles without conflict.

117789 The following summarizes the rules for the limit types:

Limit Type	Fixed Value	Minimum Acceptable Value	Maximum Acceptable Value
Standard Profile	Xs Xp == Xs (No change)	Ys Yp >= Ys (May increase the limit)	Zs Zp <= Zs (May decrease the limit)

117795 The intent is that ranges specified by limits in profiles be entirely contained within the
 117796 corresponding ranges of the base standard or standards being profiled, and that the unlimited
 117797 end of a range in a base standard must remain unlimited in any profile of that standard.

117798 Thus, the fixed `_POSIX_*` limits are constants and must not be changed by a profile. The variable
 117799 counterparts (typically without the leading `_POSIX_`) can be changed but still remain
 117800 semantically the same; that is, they still allow implementation values to vary as long as they
 117801 meet the requirements for that value (be it a minimum or maximum).

117802 Where a profile does not provide a feature upon which a limit is based, the limit is not relevant.
 117803 Applications written to that profile should be written to operate independently of the value of
 117804 the limit.

117805 An example which has previously allowed implementations to support both the base standard
 117806 and two other profiles in a compatible manner follows:

```
117807 Base standard (POSIX.1-1996): _POSIX_CHILD_MAX 6
117808 Base standard: CHILD_MAX minimum maximum _POSIX_CHILD_MAX
117809 FIPS profile/SUSv2 CHILD_MAX 25 (minimum maximum)
```

117810 **Another example:**

```
117811 Base standard (POSIX.1-1996): _POSIX_NGROUPS_MAX 0
117812 Base standard: NGROUPS_MAX minimum maximum _POSIX_NGROUP_MAX
117813 FIPS profile/SUSv2 NGROUPS_MAX 8
```

117814 A profile may lower a minimum maximum below the equivalent `_POSIX` value:

```
117815 Base standard: _POSIX_foo_MAX Z
117816 Base standard: foo_MAX _POSIX_foo_MAX
117817 profile standard : foo_MAX X (X can be less than, equal to,
117818 or greater than _POSIX_foo_MAX)
```

117819 In this case an implementation conforming to the profile may not conform to the base standard,
 117820 but an implementation to the base standard will conform to the profile.

117821 *A.2.1.6 Options*

117822 The final subsections within *Implementation Conformance* list the core options within
 117823 POSIX.1-2017. This includes both options for the System Interfaces volume of POSIX.1-2017 and
 117824 the Shell and Utilities volume of POSIX.1-2017.

117825 A.2.2 Application Conformance

117826 These definitions guide users or adapters of applications in determining on which
117827 implementations an application will run and how much adaptation would be required to make
117828 it run on others. These definitions are modeled after related ones in the ISO C standard.

117829 POSIX.1 occasionally uses the expressions “portable application” or “conforming application”.
117830 As they are used, these are synonyms for any of these terms. The differences between the classes
117831 of application conformance relate to the requirements for other standards, the options supported
117832 (such as the XSI option) or, in the case of the Conforming POSIX.1 Application Using Extensions,
117833 to implementation extensions. When one of the less explicit expressions is used, it should be
117834 apparent from the context of the discussion which of the more explicit names is appropriate

117835 A.2.2.1 Strictly Conforming POSIX Application

117836 This definition is analogous to that of an ISO C standard “conforming program”.

117837 The major difference between a Strictly Conforming POSIX Application and an ISO C standard
117838 strictly conforming program is that the latter is not allowed to use features of POSIX that are not
117839 in the ISO C standard.

117840 A.2.2.2 Conforming POSIX Application

117841 Examples of <National Bodies> include ANSI, BSI, and AFNOR.

117842 A.2.2.3 Conforming POSIX Application Using Extensions

117843 Due to possible requirements for configuration or implementation characteristics in excess of the
117844 specifications in <limits.h> or related to the hardware (such as array size or file space), not
117845 every Conforming POSIX Application Using Extensions will run on every conforming
117846 implementation.

117847 A.2.2.4 Strictly Conforming XSI Application

117848 This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX
117849 Application, with the addition of the facilities and functionality included in the XSI option.

117850 A.2.2.5 Conforming XSI Application Using Extensions

117851 Such applications may use extensions beyond the facilities defined by POSIX.1-2017 including
117852 the XSI option, but need to document the additional requirements.

117853 A.2.3 Language-Dependent Services for the C Programming Language

117854 POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and
117855 utilities, and a C binding for that specification. Efforts had been previously undertaken to
117856 generate a language-independent specification; however, that had failed, and the fact that the
117857 ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a
117858 necessary and workable situation.

117859 A.2.4 Other Language-Related Specifications

117860 There is no additional rationale provided for this section.

117861 A.3 Definitions

117862 The definitions in this section are stated so that they can be used as exact substitutes for the
117863 terms in text. They should not contain requirements or cross-references to sections within
117864 POSIX.1-2017; that is accomplished by using an informative note. In addition, the term should
117865 not be included in its own definition. Where requirements or descriptions need to be addressed
117866 but cannot be included in the definitions, due to not meeting the above criteria, these occur in
117867 the General Concepts chapter.

117868 In this version, the definitions have been reworked extensively to meet style requirements and to
117869 include terms from the base documents (see the Scope).

117870 Many of these definitions are necessarily circular, and some of the terms (such as “process”) are
117871 variants of basic computing science terms that are inherently hard to define. Where some
117872 definitions are more conceptual and contain requirements, these appear in the General Concepts
117873 chapter. Those listed in this section appear in an alphabetical glossary format of terms.

117874 Some definitions must allow extension to cover terms or facilities that are not explicitly
117875 mentioned in POSIX.1-2017. For example, the definition of “Extended Security Controls”
117876 permits implementations beyond those defined in POSIX.1-2017.

117877 Some terms in the following list of notes do not appear in POSIX.1-2017; these are marked
117878 suffixed with an asterisk (*). Many of them have been specifically excluded from POSIX.1-2017
117879 because they concern system administration, implementation, or other issues that are not
117880 specific to the programming interface. Those are marked with a reason, such as
117881 “implementation-defined”.

117882 Application

117883 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0004 [937] is applied.

117884 Appropriate Privileges

117885 One of the fundamental security problems with many historical UNIX systems has been that the
117886 privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a
117887 successful “trojan horse” attack on a privileged process defeats all security provisions.
117888 Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many
117889 historical implementations of the UNIX system, the presence of the term “appropriate
117890 privileges” in POSIX.1 may be understood as a synonym for “superuser” (UID 0). However,
117891 other systems have emerged where this is not the case and each discrete controllable action has
117892 *appropriate privileges* associated with it. Because this mechanism is implementation-defined, it
117893 must be described in the conformance document. Although that description affects several parts
117894 of POSIX.1 where the term “appropriate privilege” is used, because the term “implementation-
117895 defined” only appears here, the description of the entire mechanism and its effects on these
117896 other sections belongs in this equivalent section of the conformance document. This is especially
117897 convenient for implementations with a single mechanism that applies in all areas, since it only
117898 needs to be described once.

- 117899 **Async-Signal-Safe Function**
- 117900 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0005 [516] is applied.
- 117901 **Base Character***
- 117902 The term “Base Character” has been removed, as it was felt that the use of this term within
117903 POSIX.1-2017 was common usage English.
- 117904 **Basename**
- 117905 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0006 [653] is applied.
- 117906 **Byte**
- 117907 The restriction that a byte is now exactly eight bits was a conscious decision by the standard
117908 developers. It came about due to a combination of factors, primarily the use of the type `int8_t`
117909 within the networking functions and the alignment with the ISO/IEC 9899:1999 standard,
117910 where the `intN_t` types are now defined.
- 117911 According to the ISO/IEC 9899:1999 standard:
- 117912 The `[u]intN_t` types must be two’s complement with no padding bits and no illegal values.
- 117913 All types (apart from bit fields, which are not relevant here) must occupy an integral
117914 number of bytes.
- 117915 If a type with width W occupies B bytes with C bits per byte (C is the value of
117916 `{CHAR_BIT}`), then it has P padding bits where $P+W=B*C$.
- 117917 Therefore, for `int8_t` $P=0$, $W=8$. Since $B \geq 1$, $C \geq 8$, the only solution is $B=1$, $C=8$.
- 117918 The standard developers also felt that this was not an undue restriction for the current state-of-
117919 the-art for this version of the standard, but recognize that if industry trends continue, a wider
117920 character type may be required in the future.
- 117921 **Character**
- 117922 The term “character” is used to mean a sequence of one or more bytes representing a single
117923 graphic symbol. The deviation in the exact text of the ISO C standard definition for “byte” meets
117924 the intent of the rationale of the ISO C standard also clears up the ambiguity raised by the term
117925 “basic execution character set”. The octet-minimum requirement is a reflection of the
117926 `{CHAR_BIT}` value.
- 117927 **Child Process**
- 117928 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/3 is applied, adding the `vfork()` function
117929 to those listed.
- 117930 **Clock Tick**
- 117931 The ISO C standard defines a similar interval for use by the `clock()` function. There is no
117932 requirement that these intervals be the same. In historical implementations these intervals are
117933 different.

117934 **Command**

117935 The terms “command” and “utility” are related but have distinct meanings. Command is
 117936 defined as “a directive to a shell to perform a specific task”. The directive can be in the form of a
 117937 single utility name (for example, *ls*), or the directive can take the form of a compound command
 117938 (for example, “*ls | grep name | pr*”). A utility is a program that can be called by name
 117939 from a shell. Issuing only the name of the utility to a shell is the equivalent of a one-word
 117940 command. A utility may be invoked as a separate program that executes in a different process
 117941 than the command language interpreter, or it may be implemented as a part of the command
 117942 language interpreter. For example, the *echo* command (the directive to perform a specific task)
 117943 may be implemented such that the *echo* utility (the logic that performs the task of echoing) is in a
 117944 separate program; therefore, it is executed in a process that is different from the command
 117945 language interpreter. Conversely, the logic that performs the *echo* utility could be built into the
 117946 command language interpreter; therefore, it could execute in the same process as the command
 117947 language interpreter.

117948 The terms “tool” and “application” can be thought of as being synonymous with “utility” from
 117949 the perspective of the operating system kernel. Tools, applications, and utilities historically have
 117950 run, typically, in processes above the kernel level. Tools and utilities historically have been a part
 117951 of the operating system non-kernel code and have performed system-related functions, such as
 117952 listing directory contents, checking file systems, repairing file systems, or extracting system
 117953 status information. Applications have not generally been a part of the operating system, and
 117954 they perform non-system-related functions, such as word processing, architectural design,
 117955 mechanical design, workstation publishing, or financial analysis. Utilities have most frequently
 117956 been provided by the operating system distributor, applications by third-party software
 117957 distributors, or by the users themselves. Nevertheless, POSIX.1-2017 does not differentiate
 117958 between tools, utilities, and applications when it comes to receiving services from the system, a
 117959 shell, or the standard utilities. (For example, the *xargs* utility invokes another utility; it would be
 117960 of fairly limited usefulness if the users could not run their own applications in place of the
 117961 standard utilities.) Utilities are not applications in the sense that they are not themselves subject
 117962 to the restrictions of POSIX.1-2017 or any other standard. There is no requirement for *grep*, *stty*,
 117963 or any of the utilities defined here to be any of the classes of conforming applications.

117964 **Column Positions**

117965 In most 1-byte character sets, such as ASCII, the concept of column positions is identical to
 117966 character positions and to bytes. Therefore, it has been historically acceptable for some
 117967 implementations to describe line folding or tab stops or table column alignment in terms of
 117968 bytes or character positions. Other character sets pose complications, as they can have internal
 117969 representations longer than one octet and they can have display characters that have different
 117970 widths on the terminal screen or printer.

117971 In POSIX.1-2017 the term “column positions” has been defined to mean character (not byte) n
 117972 positions in input files (such as “column position 7 of the FORTRAN input”). Output files
 117973 describe the column position in terms of the display width of the narrowest printable character
 117974 in the character set, adjusted to fit the characteristics of the output device. It is very possible that
 117975 n column positions will not be able to hold n characters in some character sets, unless all of those
 117976 characters are of the narrowest width. It is assumed that the implementation is aware of the
 117977 width of the various characters, deriving this information from the value of *LC_CTYPE*, and
 117978 thus can determine how many column positions to allot for each character in those utilities
 117979 where it is important.

117980 The term “column position” was used instead of the more natural “column” because the latter is
 117981 frequently used in the different contexts of columns of figures, columns of table values, and so
 117982 on. Wherever confusion might result, these latter types of columns are referred to as “text

- 117983 columns".
- 117984 **Controlling Terminal**
- 117985 The question of which of possibly several special files referring to the terminal is meant is not
117986 addressed in POSIX.1. The pathname `/dev/tty` is a synonym for the controlling terminal
117987 associated with a process.
- 117988 **Device Number***
- 117989 The concept is handled in `stat()` as *ID of device*.
- 117990 **Direct I/O**
- 117991 Historically, direct I/O refers to the system bypassing intermediate buffering, but may be
117992 extended to cover implementation-defined optimizations.
- 117993 **Directory**
- 117994 The format of the directory file is implementation-defined and differs radically between
117995 System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and
117996 certain constraints on the format of the information returned by those routines are described in
117997 the `<dirent.h>` header.
- 117998 **Directory Entry**
- 117999 Throughout POSIX.1-2017, the term "link" is used (about the `link()` function, for example) in
118000 describing the objects that point to files from directories.
- 118001 **Display**
- 118002 The Shell and Utilities volume of POSIX.1-2017 assigns precise requirements for the terms
118003 "display" and "write". Some historical systems have chosen to implement certain utilities
118004 without using the traditional file descriptor model. For example, the *vi* editor might employ
118005 direct screen memory updates on a personal computer, rather than a `write()` system call. An
118006 instance of user prompting might appear in a dialog box, rather than with standard error. When
118007 the Shell and Utilities volume of POSIX.1-2017 uses the term "display", the method of
118008 outputting to the terminal is unspecified; many historical implementations use *termcap* or
118009 *terminfo*, but this is not a requirement. The term "write" is used when the Shell and Utilities
118010 volume of POSIX.1-2017 mandates that a file descriptor be used and that the output can be
118011 redirected. However, it is assumed that when the writing is directly to the terminal (it has not
118012 been redirected elsewhere), there is no practical way for a user or test suite to determine whether
118013 a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the
118014 redirection case and the implementation is free to use any method when the output is not
118015 redirected. The verb *write* is used almost exclusively, with the very few exceptions of those
118016 utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.
- 118017 **Dot**
- 118018 The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory
118019 filename from a period or a decimal point.

118020 Dot-Dot

118021 Historical implementations permit the use of these filenames without their special meanings.
118022 Such use precludes any meaningful use of these filenames by a Conforming POSIX.1
118023 Application. Therefore, such use is considered an extension, the use of which makes an
118024 implementation non-conforming; see also [Section A.4.13](#) (on page 3516).

118025 Epoch

118026 Historically, the origin of UNIX system time was referred to as ``00:00:00 GMT, January 1, 1970''.
118027 Greenwich Mean Time is actually not a term acknowledged by the international standards
118028 community; therefore, this term, ``Epoch'', is used to abbreviate the reference to the actual
118029 standard, Coordinated Universal Time.

118030 FIFO Special File

118031 See [Pipe](#) (on page 3500).

118032 File

118033 It is permissible for an implementation-defined file type to be non-readable or non-writable.

118034 File Classes

118035 These classes correspond to the historical sets of permission bits. The classes are general to
118036 allow implementations flexibility in expanding the access mechanism for more stringent security
118037 environments. Note that a process is in one and only one class, so there is no ambiguity.

118038 File Mode

118039 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0007 [834] is applied.

118040 Filename

118041 Filenames are sequences of bytes, not sequences of characters. The only bytes that this standard
118042 says cannot appear in any filename are the slash byte and the null byte. This is a side-effect of
118043 the fact that no conforming implementations of the standard currently provide a way to pass
118044 information specifying the locale associated with strings passed between user-level applications
118045 and the kernel. This decision could be revisited if implementations develop a way to associate a
118046 locale with the strings passed between kernel space and user space.

118047 Implementations may add other restrictions to the byte sequences allowed in filenames except
118048 that any filename consisting of no more than {NAME_MAX} bytes from the set of characters in
118049 the portable filename character set must be allowed.

118050 See [Pathname](#) (on page 3500).

118051 File System

118052 Historically, the meaning of this term has been overloaded with two meanings: that of the
118053 complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file
118054 system. POSIX.1 uses the term ``file system'' in the second sense, except that it is limited to the
118055 scope of a process (and root directory of a process). This usage also clarifies the domain in which
118056 a file serial number is unique.

118057 Graphic Character

118058 This definition is made available for those definitions (in particular, *TZ*) which must exclude
118059 control characters.

118060 Group Database

118061 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/4 is applied, removing the words “of
118062 implementation-defined format”. See [User Database](#) (on page 3510).

118063 Group File*

118064 Implementation-defined; see [User Database](#) (on page 3510).

118065 Group ID

118066 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0008 [511] is applied.

118067 Group Name

118068 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0009 [584] is applied.

118069 Historical Implementations*

118070 This refers to previously existing implementations of programming interfaces and operating
118071 systems that are related to the interface specified by POSIX.1.

118072 Hosted Implementation*

118073 This refers to a POSIX.1 implementation that is accomplished through interfaces from the
118074 POSIX.1 services to some alternate form of operating system kernel services. Note that the line
118075 between a hosted implementation and a native implementation is blurred, since most
118076 implementations will provide some services directly from the kernel and others through some
118077 indirect path. (For example, *fopen()* might use *open()*; or *mkfifo()* might use *mknod()*.) There is
118078 no necessary relationship between the type of implementation and its correctness, performance,
118079 and/or reliability.

118080 Implementation*

118081 This term is generally used instead of its synonym, “system”, to emphasize the consequences of
118082 decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1
118083 were allowed, this usage would not have occurred.

118084 The term “specific implementation” is sometimes used as a synonym for “implementation”.
118085 This should not be interpreted too narrowly; both terms can represent a relatively broad group
118086 of systems. For example, a hardware vendor could market a very wide selection of systems that
118087 all used the same instruction set, with some systems desktop models and others large multi-user
118088 minicomputers. This wide range would probably share a common POSIX.1 operating system,
118089 allowing an application compiled for one to be used on any of the others; this is a [*specific*]
118090 *implementation*. However, such a wide range of machines probably has some differences
118091 between the models. Some may have different clock rates, different file systems, different
118092 resource limits, different network connections, and so on, depending on their sizes or intended
118093 usages. Even on two identical machines, the system administrators may configure them
118094 differently. Each of these different systems is known by the term “a specific instance of a specific
118095 implementation”. This term is only used in the portions of POSIX.1 dealing with runtime
118096 queries: *sysconf()* and *pathconf()*.

118097 Incomplete Pathname*

118098 Absolute pathname has been adequately defined.

118099 Job Control

118100 In order to understand the job control facilities in POSIX.1 it is useful to understand how they
118101 are used by a job control-cognizant shell to create the user interface effect of job control.

118102 While the job control facilities supplied by POSIX.1 can, in theory, support different types of
118103 interactive job control interfaces supplied by different types of shells, there was historically one
118104 particular interface that was most common when the standard was originally developed
118105 (provided by BSD C Shell).

118106 This discussion describes that interface as a means of illustrating how the POSIX.1 job control
118107 facilities can be used.

118108 Job control allows users to selectively stop (suspend) the execution of processes and continue
118109 (resume) their execution at a later point. The user typically employs this facility via the
118110 interactive interface jointly supplied by the terminal I/O driver and a command interpreter
118111 (shell).

118112 The user can launch jobs (command pipelines) in either the foreground or background. When
118113 launched in the foreground, the shell waits for the job to complete before prompting for
118114 additional commands. When launched in the background, the shell does not wait, but
118115 immediately prompts for new commands.

118116 If the user launches a job in the foreground and subsequently regrets this, the user can type the
118117 suspend character (typically set to <control>-Z), which causes the foreground job to stop and the
118118 shell to begin prompting for new commands. The stopped job can be continued by the user (via
118119 special shell commands) either as a foreground job or as a background job. Background jobs can
118120 also be moved into the foreground via shell commands.

118121 If a background job attempts to access the login terminal (controlling terminal), it is stopped by
118122 the terminal driver and the shell is notified, which, in turn, notifies the user. (Terminal access
118123 includes *read()* and certain terminal control functions, and conditionally includes *write()*.) The
118124 user can continue the stopped job in the foreground, thus allowing the terminal access to
118125 succeed in an orderly fashion. After the terminal access succeeds, the user can optionally move
118126 the job into the background via the suspend character and shell commands.

118127 Implementing Job Control Shells

118128 The interactive interface described previously can be accomplished using the POSIX.1 job
118129 control facilities in the following way.

118130 The key feature necessary to provide job control is a way to group processes into jobs. This
118131 grouping is necessary in order to direct signals to a single job and also to identify which job is in
118132 the foreground. (There is at most one job that is in the foreground on any controlling terminal at
118133 a time.)

118134 The concept of process groups is used to provide this grouping. The shell places each job in a
118135 separate process group via the *setpgid()* function. To do this, the *setpgid()* function is invoked by
118136 the shell for each process in the job. It is actually useful to invoke *setpgid()* twice for each
118137 process: once in the child process, after calling *fork()* to create the process, but before calling one
118138 of the *exec* family of functions to begin execution of the program, and once in the parent shell
118139 process, after calling *fork()* to create the child. The redundant invocation avoids a race condition
118140 by ensuring that the child process is placed into the new process group before either the parent
118141 or the child relies on this being the case. The process group ID for the job is selected by the shell
118142 to be equal to the process ID of one of the processes in the job. Some shells choose to make one

118143 process in the job be the parent of the other processes in the job (if any). Other shells (for
118144 example, the C Shell) choose to make themselves the parent of all processes in the pipeline (job).
118145 In order to support this latter case, the *setpgid()* function accepts a process group ID parameter
118146 since the correct process group ID cannot be inherited from the shell. The shell itself is
118147 considered to be a job and is the sole process in its own process group.

118148 The shell also controls which job is currently in the foreground. A foreground and background
118149 job differ in two ways: the shell waits for a foreground command to complete (or stop) before
118150 continuing to read new commands, and the terminal I/O driver inhibits terminal access by
118151 background jobs (causing the processes to stop). Thus, the shell must work cooperatively with
118152 the terminal I/O driver and have a common understanding of which job is currently in the
118153 foreground. It is the user who decides which command should be currently in the foreground,
118154 and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O
118155 driver via the *tcsetpgrp()* function. This indicates to the terminal I/O driver the process group ID
118156 of the foreground process group (job). When the current foreground job either stops or
118157 terminates, the shell places itself in the foreground via *tcsetpgrp()* before prompting for
118158 additional commands. Note that when a job is created the new process group begins as a
118159 background process group. It requires an explicit act of the shell via *tcsetpgrp()* to move a
118160 process group (job) into the foreground.

118161 When a process in a job stops or terminates, its parent (for example, the shell) receives
118162 synchronous notification by calling the *waitpid()* function with the WUNTRACED flag set.
118163 Asynchronous notification is also provided when the parent establishes a signal handler for
118164 SIGCHLD and does not specify the SA_NOCLDSTOP flag. Usually all processes in a job stop as
118165 a unit since the terminal I/O driver always sends job control stop signals to all processes in the
118166 process group.

118167 To continue a stopped job, the shell sends the SIGCONT signal to the process group of the job. In
118168 addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp()* to place the
118169 job in the foreground before sending SIGCONT. Otherwise, the shell leaves itself in the
118170 foreground and reads additional commands.

118171 There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the
118172 typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write()*
118173 functions without stopping. The same effect can be achieved on a per-process basis by having a
118174 process set the signal action for SIGTTOU to SIG_IGN.

118175 Note that the terms “job” and “process group” can be used interchangeably. A login session that
118176 is not using the job control facilities can be thought of as a large collection of processes that are
118177 all in the same job (process group). Such a login session may have a partial distinction between
118178 foreground and background processes; that is, the shell may choose to wait for some processes
118179 before continuing to read new commands and may not wait for other processes. However, the
118180 terminal I/O driver will consider all these processes to be in the foreground since they are all
118181 members of the same process group.

118182 In addition to the basic job control operations already mentioned, a job control-cognizant shell
118183 needs to perform the following actions.

118184 When a foreground (not background) job stops, the shell must sample and remember the current
118185 terminal settings so that it can restore them later when it continues the stopped job in the
118186 foreground (via the *tcgetattr()* and *tcsetattr()* functions).

118187 Because a shell itself can be spawned from a shell, it must take special action to ensure that
118188 subshells interact well with their parent shells.

118189 A subshell can be spawned to perform an interactive function (prompting the terminal for
118190 commands) or a non-interactive function (reading commands from a file). When operating non-

118191 interactively, the job control shell will refrain from performing the job control-specific actions
118192 described above. It will behave as a shell that does not support job control. For example, all jobs
118193 will be left in the same process group as the shell, which itself remains in the process group
118194 established for it by its parent. This allows the shell and its children to be treated as a single job
118195 by a parent shell, and they can be affected as a unit by terminal keyboard signals.

118196 An interactive subshell can be spawned from another job control-cognizant shell in either the
118197 foreground or background. (For example, from the C Shell, the user can execute the command,
118198 `csch &`.) Before the subshell activates job control by calling `setpgid()` to place itself in its own
118199 process group and `tcsetgrp()` to place its new process group in the foreground, it needs to
118200 ensure that it has already been placed in the foreground by its parent. (Otherwise, there could
118201 be multiple job control shells that simultaneously attempt to control mediation of the terminal.)
118202 To determine this, the shell retrieves its own process group via `getpgrp()` and the process group
118203 of the current foreground job via `tcgetpgrp()`. If these are not equal, the shell sends SIGTTIN to
118204 its own process group, causing itself to stop. When continued later by its parent, the shell
118205 repeats the process group check. When the process groups finally match, the shell is in the
118206 foreground and it can proceed to take control. After this point, the shell ignores all the job
118207 control stop signals so that it does not inadvertently stop itself.

118208 *Implementing Job Control Applications*

118209 Most applications do not need to be aware of job control signals and operations; the intuitively
118210 correct behavior happens by default. However, sometimes an application can inadvertently
118211 interfere with normal job control processing, or an application may choose to overtly effect job
118212 control in cooperation with normal shell procedures.

118213 An application can inadvertently subvert job control processing by “blindly” altering the
118214 handling of signals. A common application error is to learn how many signals the system
118215 supports and to ignore or catch them all. Such an application makes the assumption that it does
118216 not know what this signal is, but knows the right handling action for it. The system may
118217 initialize the handling of job control stop signals so that they are being ignored. This allows
118218 shells that do not support job control to inherit and propagate these settings and hence to be
118219 immune to stop signals. A job control shell will set the handling to the default action and
118220 propagate this, allowing processes to stop. In doing so, the job control shell is taking
118221 responsibility for restarting the stopped applications. If an application wishes to catch the stop
118222 signals itself, it should first determine their inherited handling states. If a stop signal is being
118223 ignored, the application should continue to ignore it. This is directly analogous to the
118224 recommended handling of SIGINT described in the referenced UNIX Programmer’s Manual.

118225 If an application is reading the terminal and has disabled the interpretation of special characters
118226 (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend
118227 character is typed. Such an application can simulate the effect of the suspend character by
118228 recognizing it and sending SIGTSTP to its process group as the terminal driver would have
118229 done. Note that the signal is sent to the process group, not just to the application itself; this
118230 ensures that other processes in the job also stop. (Note also that other processes in the job could
118231 be children, siblings, or even ancestors.) Applications should not assume that the suspend
118232 character is `<control>-Z` (or any particular value); they should retrieve the current setting at
118233 startup.

118234 *Implementing Job Control Systems*

118235 The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD
 118236 programmatic interface with only minimal changes to resolve syntactic or semantic conflicts
 118237 with System V or to close recognized security holes. The goal was to maximize the ease of
 118238 providing both conforming implementations and Conforming POSIX.1 Applications.

118239 It is only useful for a process to be affected by job control signals if it is the descendant of a job
 118240 control shell. Otherwise, there will be nothing that continues the stopped process.

118241 POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that
 118242 is, by a controlling process terminating). 4.2 BSD uses the *vhangup()* function to prevent any
 118243 access to the controlling terminal through file descriptors opened prior to logout. System V does
 118244 not prevent controlling terminal access through file descriptors opened prior to logout (except
 118245 for the case of the special file, */dev/tty*). Some implementations choose to make processes
 118246 immune from job control after logout (that is, such processes are always treated as if in the
 118247 foreground); other implementations continue to enforce foreground/background checks after
 118248 logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the
 118249 controlling terminal after logout since such access is unreliable. If an implementation chooses to
 118250 deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain
 118251 type of behavior (see [Controlling Terminal](#), on page 3491).

118252 **Kernel***

118253 See [System Call*](#) (on page 3508).

118254 **Library Routine***

118255 See [System Call*](#) (on page 3508).

118256 **Live Process**

118257 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0010 [690] is applied.

118258 **Logical Device***

118259 Implementation-defined.

118260 **Map**

118261 The definition of map is included to clarify the usage of mapped pages in the description of the
 118262 behavior of process memory locking.

118263 **Memory-Resident**

118264 The term “memory-resident” is historically understood to mean that the so-called resident pages
 118265 are actually present in the physical memory of the computer system and are immune from
 118266 swapping, paging, copy-on-write faults, and so on. This is the actual intent of POSIX.1-2017 in
 118267 the process memory locking section for implementations where this is logical. But for some
 118268 implementations †primarily mainframes ‡actually locking pages into primary storage is not
 118269 advantageous to other system objectives, such as maximizing throughput. For such
 118270 implementations, memory locking is a “hint” to the implementation that the application wishes
 118271 to avoid situations that would cause long latencies in accessing memory. Furthermore, there are
 118272 other implementation-defined issues with minimizing memory access latencies that “memory
 118273 residency” does not address—such as MMU reload faults. The definition attempts to
 118274 accommodate various implementations while allowing conforming applications to specify to the
 118275 implementation that they want or need the best memory access times that the implementation

118276 can provide.

118277 **Memory Object***

118278 The term “memory object” usually implies shared memory. If the object is the same as a
118279 filename in the file system name space of the implementation, it is expected that the data written
118280 into the memory object be preserved on disk. A memory object may also apply to a physical
118281 device on an implementation. In this case, writes to the memory object are sent to the controller
118282 for the device and reads result in control registers being returned.

118283 **Mount Point***

118284 The directory on which a “mounted file system” is mounted. This term, like *mount()* and
118285 *umount()*, was not included because it was implementation-defined.

118286 **Mounted File System***

118287 See [File System](#) (on page 3492).

118288 **Multi-Threaded Library**

118289 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

118290 **Multi-Threaded Process**

118291 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

118292 **Multi-Threaded Program**

118293 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.

118294 **Name**

118295 There are no explicit limits in POSIX.1-2017 on the sizes of names, words (see the definition of
118296 word in the Base Definitions volume of POSIX.1-2017), lines, or other objects. However, other
118297 implicit limits do apply: shell script lines produced by many of the standard utilities cannot
118298 exceed {LINE_MAX} and the sum of exported variables comes under the {ARG_MAX} limit.
118299 Historical shells dynamically allocate memory for names and words and parse incoming lines a
118300 character at a time. Lines cannot have an arbitrary {LINE_MAX} limit because of historical
118301 practice, such as makefiles, where *make* removes the <newline> characters associated with the
118302 commands for a target and presents the shell with one very long line. The text on INPUT FILES
118303 in XCU [Section 1.4](#) (on page 2336) does allow a shell to run out of memory, but it cannot have
118304 arbitrary programming limits.

118305 **Native Implementation***

118306 This refers to an implementation of POSIX.1 that interfaces directly to an operating system
118307 kernel; see also *hosted implementation*. A similar concept is a native UNIX system, which would
118308 be a kernel derived from one of the original UNIX system products.

118309 Nice Value

118310 This definition is not intended to suggest that all processes in a system have priorities that are
118311 comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of
118312 a single underlying priority for all scheduling policies problematic. Some implementations may
118313 implement the features related to *nice* to affect all processes on the system, others to affect just
118314 the general time-sharing activities implied by POSIX.1-2017, and others may have no effect at all.
118315 Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of
118316 implementation strategies is possible.

118317 Open File Description

118318 An “open file description”, as it is currently named, describes how a file is being accessed. What
118319 is currently called a “file descriptor” is actually just an identifier or “handle”; it does not
118320 actually describe anything.

118321 The following alternate names were discussed:

118322 For “open file description”:

118323 “open instance”, “file access description”, “open file information”, and “file access
118324 information”.

118325 For “file descriptor”:

118326 “file handle”, “file number” (cf., *fileno()*). Some historical implementations use the term
118327 “file table entry”.

118328 Orphaned Process Group

118329 Historical implementations have a concept of an orphaned process, which is a process whose
118330 parent process has exited. When job control is in use, it is necessary to prevent processes from
118331 being stopped in response to interactions with the terminal after they no longer are controlled by
118332 a job control-cognizant program. Because signals generated by the terminal are sent to a process
118333 group and not to individual processes, and because a signal may be provoked by a process that
118334 is not orphaned, but sent to another process that is orphaned, it is necessary to define an
118335 orphaned process group. The definition assumes that a process group will be manipulated as a
118336 group and that the job control-cognizant process controlling the group is outside of the group
118337 and is the parent of at least one process in the group (so that state changes may be reported via
118338 *waitpid()*). Therefore, a group is considered to be controlled as long as at least one process in the
118339 group has a parent that is outside of the process group, but within the session.

118340 This definition of orphaned process groups ensures that a session leader’s process group is
118341 always considered to be orphaned, and thus it is prevented from stopping in response to
118342 terminal signals.

118343 Page

118344 The term “page” is defined to support the description of the behavior of memory mapping for
118345 shared memory and memory mapped files, and the description of the behavior of process
118346 memory locking. It is not intended to imply that shared memory/file mapping and memory
118347 locking are applicable only to “paged” architectures. For the purposes of POSIX.1-2017,
118348 whatever the granularity on which an architecture supports mapping or locking, this is
118349 considered to be a “page”. If an architecture cannot support the memory mapping or locking
118350 functions specified by POSIX.1-2017 on any granularity, then these options will not be
118351 implemented on the architecture.

118352 Pathname

118353 Pathnames historically allowed all bytes except for the <slash> and <NUL> characters. For
118354 compatibility with existing file systems, this usage is maintained throughout the standard by
118355 noting that a pathname need not be a valid character string in all locales. However, the
118356 properties of the portable filename character set are such that a pathname using only those
118357 characters and the <slash> is portable in all locales as a character string.

118358 Passwd File*

118359 Implementation-defined; see [User Database](#) (on page 3510).

118360 Parent Directory

118361 There may be more than one directory entry pointing to a given directory in some
118362 implementations. The wording here identifies that exactly one of those is the parent directory. In
118363 pathname resolution, dot-dot is identified as the way that the unique directory is identified.
118364 (That is, the parent directory is the one to which dot-dot points.) In the case of a remote file
118365 system, if the same file system is mounted several times, it would appear as if they were distinct
118366 file systems (with interesting synchronization properties).

118367 Pipe

118368 It proved convenient to define a pipe as a special case of a FIFO, even though historically the
118369 latter was not introduced until System III and does not exist at all in 4.3 BSD.

118370 Portable Filename

118371 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0012 [584] is applied.

118372 Portable Filename Character Set

118373 The encoding of this character set is not specified—specifically, ASCII is not required. But the
118374 implementation must provide a unique character code for each of the printable graphics
118375 specified by POSIX.1; see also [Section A.4.7](#) (on page 3512).

118376 Situations where characters beyond the portable filename character set (or historically ASCII or
118377 the ISO/IEC 646:1991 standard) would be used (in a context where the portable filename
118378 character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be
118379 common. Although such a situation renders the use technically non-compliant, mutual
118380 agreement among the users of an extended character set will make such use portable between
118381 those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1.
118382 (Making it required would eliminate too many possible systems, as even those systems using the
118383 ISO/IEC 646:1991 standard as a base character set extend their character sets for Western
118384 Europe and the rest of the world in different ways.)

118385 Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is
118386 not required or where mutual agreement is obtained. It has been suggested that in several places
118387 “should” be used instead of “shall”. Because (in the worst case) use of any character beyond the
118388 portable filename character set would render the program or data not portable to all possible
118389 systems, no extensions are permitted in this context.

118390 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0013 [584] is applied.

- 118391 **Process**
- 118392 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.
- 118393 **Process Lifetime**
- 118394 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/5 is applied, adding *fork()*, *posix_spawn()*,
118395 *posix_spawnnp()*, and *vfork()* to the list of functions.
- 118396 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.
- 118397 **Process Termination**
- 118398 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/6 is applied, rewording the definition to
118399 address the “passive exit” on termination of the last thread or the *_Exit()* function.
- 118400 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0014 [690] is applied.
- 118401 **Regular File**
- 118402 POSIX.1 does not intend to preclude the addition of structuring data (for example, record
118403 lengths) in the file, as long as such data is not visible to an application that uses the features
118404 described in POSIX.1.
- 118405 **Root Directory**
- 118406 This definition permits the operation of *chroot()*, even though that function is not in POSIX.1; see
118407 also [Section A.4.6](#) (on page 3512).
- 118408 **Root File System***
- 118409 Implementation-defined.
- 118410 **Root of a File System***
- 118411 Implementation-defined; see [Mount Point*](#) (on page 3498).
- 118412 **Signal**
- 118413 The definition implies a double meaning for the term. Although a signal is an event, common
118414 usage implies that a signal is an identifier of the class of event.
- 118415 **Single-Threaded Process**
- 118416 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.
- 118417 **Single-Threaded Program**
- 118418 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0011 [625] is applied.
- 118419 **Source Code**
- 118420 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0015 [896] is applied.

118421 **Superuser***

118422 This concept, with great historical significance to UNIX system users, has been replaced with the
118423 notion of appropriate privileges.

118424 **Supplementary Group ID**

118425 The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The
118426 definition of supplementary group ID explicitly permits the effective group ID to be included in
118427 the set, but wording in the description of the *setuid()* and *setgid()* functions states: “Any
118428 supplementary group IDs of the calling process remain unchanged by these function calls”.
118429 In the case of *setgid()* this contradicts that definition. In addition, some felt that the unspecified
118430 behavior in the definition of supplementary group IDs adds unnecessary portability problems.
118431 The standard developers considered several solutions to this problem:

- 118432 1. Reword the description of *setgid()* to permit it to change the supplementary group IDs to
118433 reflect the new effective group ID. A problem with this is that it adds more “may”s to the
118434 wording and does not address the portability problems of this optional behavior.
- 118435 2. Mandate the inclusion of the effective group ID in the supplementary set (giving
118436 {NGROUPS_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that
118437 system, the effective group ID is the first element of the array of supplementary group
118438 IDs (there is no separate copy stored, and changes to the effective group ID are made only
118439 in the supplementary group set). By convention, the initial value of the effective group ID
118440 is duplicated elsewhere in the array so that the initial value is not lost when executing a
118441 set-group-ID program.
- 118442 3. Change the definition of supplementary group ID to exclude the effective group ID and
118443 specify that the effective group ID does not change the set of supplementary group IDs.
118444 This is the behavior of 4.2 BSD, 4.3 BSD, and System V Release 4.
- 118445 4. Change the definition of supplementary group ID to exclude the effective group ID, and
118446 require that *getgroups()* return the union of the effective group ID and the supplementary
118447 group IDs.
- 118448 5. Change the definition of {NGROUPS_MAX} to be one more than the number of
118449 supplementary group IDs, so it continues to be the number of values returned by
118450 *getgroups()* and existing applications continue to work. This alternative is effectively the
118451 same as the second (and might actually have the same implementation).

118452 The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to
118453 the set of supplementary group IDs, and it is implementation-defined whether *getgroups()*
118454 returns this. If the effective group ID is returned with the set of supplementary group IDs, then
118455 all changes to the effective group ID affect the supplementary group set returned by *getgroups()*.
118456 It is permissible to eliminate duplicates from the list returned by *getgroups()*. However, if a
118457 group ID is contained in the set of supplementary group IDs, setting the group ID to that value
118458 and then to a different value should not remove that value from the supplementary group IDs.

118459 The definition of supplementary group IDs has been changed to not include the effective group
118460 ID. This simplifies permanent rationale and makes the relevant functions easier to understand.
118461 The *getgroups()* function has been modified so that it can, on an implementation-defined basis,
118462 return the effective group ID. By making this change, functions that modify the effective group
118463 ID do not need to discuss adding to the supplementary group list; the only view into the
118464 supplementary group list that the application developer has is through the *getgroups()* function.

118465 Symbolic Constant

118466 Earlier versions of this standard used a variety of terms other than “macro” for many of the
 118467 constants defined in headers, and it was not clear in which of these cases they were required to
 118468 be macros or not, or to be pre-processor constants (i.e., usable in `#if`) or not. In cases where the
 118469 symbols had a reserved prefix or suffix, there was often inconsistency between whether the
 118470 prefix/suffix was reserved only for macros or for any use, and whether the term “macro” or a
 118471 different term was used in the descriptions of the symbols. There were also some unintentional
 118472 differences from the ISO C standard.

118473 One of the most commonly used terms was “symbolic constant”. This has now been designated
 118474 as the default term to be used wherever appropriate, and a formal definition of the term has
 118475 been added giving the exact requirements for symbols that are described as symbolic constants.

118476 The standard developers have performed a major rationalization of the header descriptions of
 118477 symbols with constant values according to the following policy:

118478 Where symbols are from the ISO C standard, the wording from the ISO C standard (or
 118479 equivalent, in cases where the exact wording is not appropriate) is used to describe them.

118480 For all other constants, the first choice is to use “symbolic constant” when the
 118481 requirements for the symbol are a reasonably close fit with those of the term.

118482 The description of the symbol can override individual requirements for symbolic
 118483 constants; e.g., to specify a non-integer type, or to add a requirement that the symbol is
 118484 usable in `#if` preprocessor directives.

118485 When neither of the above apply, the exact requirements are stated in the description.
 118486 (Note that macros are not required to be usable in `#if`, or even to expand to constant
 118487 expressions, unless explicitly stated.)

118488 In cases where there is a reserved prefix or suffix, if the symbol(s) with that prefix/suffix
 118489 are from the ISO C standard and are required to be macros, or if the symbol is required to
 118490 be usable in `#if`, then the prefix/suffix is reserved for use only as macros. If the symbol(s)
 118491 are “symbolic constants” and not required to be usable in `#if`, the prefix/suffix is reserved
 118492 for any use except in a few special cases.

118493 Where a constant is required to be a macro but is also allowed to be another type of constant
 118494 such as an enumeration constant, on implementations which do define it as another type of
 118495 constant the macro is typically defined as follows:

```
118496 #define macro_name macro_name
```

118497 This allows applications to use `#ifdef`, etc. to determine whether the macro is defined, but the
 118498 macro is not usable in `#if` preprocessor directives because the preprocessor will treat the
 118499 unexpanded word `macro_name` as having the value zero.

118500 Symbolic Link

118501 Earlier versions of this standard did not require symbolic links to have attributes such as
 118502 ownership and a file serial number. This was because the 4.4 BSD implementation did not have
 118503 them, and it was expected that other implementations may wish to do the same. However,
 118504 experience with 4.4 BSD has shown that symbolic links implemented in this way cause problems
 118505 for users and application developers, and later BSD systems have reverted to using `inodes` to
 118506 implement symbolic links. Allowing `no-inode` symbolic links also caused problems in the
 118507 standard. For example, leaving the `st_ino` value for symbolic links unspecified meant that the
 118508 common technique of comparing the `st_dev` and `st_ino` values for two pathnames to see if they
 118509 refer to the same file could only be used with `stat()` in conforming applications and not with
 118510 `lstat()`. The standard now requires symbolic links to have meaningful values for the same `struct`

118511 **stat** fields as regular files, except for the file mode bits in *st_mode*. Historically, the file mode bits
118512 were unused (the contents of a symbolic link could always be read), but implementations
118513 differed as to whether the file mode bits (as returned in *st_mode* or reported by *ls -l*) were set
118514 according to the *umask* or just to a fixed value such as 0777. Accordingly, the standard requires
118515 the file mode bits to be ignored by *readlink()* and when a symbolic link is followed during
118516 pathname resolution, but leaves the corresponding part of the value returned in *st_mode*
118517 unspecified.

118518 Historical implementations were followed when determining which interfaces should apply to
118519 symbolic links. Interfaces that historically followed symbolic links include *chmod()*, *stat()*, and
118520 *utime()*. Interfaces that historically did not follow symbolic links include *lstat()*, *rename()*,
118521 *remove()*, *rmdir()*, and *unlink()*. For *chown()* and *link()*, historical implementations differed.
118522 POSIX.1-2017 inherited the *lchown()* function from the Single UNIX Specification, Version 2, and
118523 therefore requires *chown()* to follow symbolic links. Earlier versions of this standard required
118524 *link()* to follow symbolic links, but with the addition of the *linkat()* function (which has a flag to
118525 indicate whether to follow symbolic links), both behaviors are now allowed for *link()*.

118526 When the final component of a pathname is a symbolic link, the standard requires that a trailing
118527 <slash> causes the link to be followed. This is the behavior of historical implementations. For
118528 example, for */a/b* and */a/b/*, if */a/b* is a symbolic link to a directory, then */a/b* refers to the
118529 symbolic link, and */a/b/* refers to the directory to which the symbolic link points.

118530 Because a symbolic link and its referenced object coexist in the file system name space, confusion
118531 can arise in distinguishing between the link itself and the referenced object. Historically, utilities
118532 and system calls have adopted their own link following conventions in a somewhat *ad hoc*
118533 fashion. Rules for a uniform approach are outlined here, although historical practice has been
118534 adhered to as much as was possible. To promote consistent system use, user-written utilities are
118535 encouraged to follow these same rules.

118536 Symbolic links are handled either by operating on the link itself, or by operating on the object
118537 referenced by the link. In the latter case, an application or system call is said to “follow” the link.
118538 Symbolic links may reference other symbolic links, in which case links are dereferenced until an
118539 object that is not a symbolic link is found, a symbolic link that references a file that does not exist
118540 is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on
118541 the number of symbolic links that they will dereference before declaring it an error.)

118542 There are four domains for which default symbolic link policy is established in a system. In
118543 almost all cases, there are utility options that override this default behavior. The four domains
118544 are as follows:

- 118545 1. Symbolic links specified to system calls that take pathname arguments
- 118546 2. Symbolic links specified as command line pathname arguments to utilities that are not
118547 performing a traversal of a file hierarchy
- 118548 3. Symbolic links referencing files not of type directory, specified to utilities that are
118549 performing a traversal of a file hierarchy
- 118550 4. Symbolic links referencing files of type directory, specified to utilities that are performing
118551 a traversal of a file hierarchy

118552 *First Domain*

118553 The first domain is considered in earlier rationale.

118554 *Second Domain*

118555 The reason this category is restricted to utilities that are not traversing the file hierarchy is that
 118556 some standard utilities take an option that specifies a hierarchical traversal, but by default
 118557 operate on the arguments themselves. Generally, users specifying the option for a file hierarchy
 118558 traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which
 118559 may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a
 118560 different operation from the same command with the **-R** option specified. In this example, the
 118561 behavior of the command *chown owner file* is described here, while the behavior of the command
 118562 *chown -R owner file* is described in the third and fourth domains.

118563 The general rule is that the utilities in this category follow symbolic links named as arguments.

118564 Exceptions in the second domain are:

118565 The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively
 118566 attempt to rename or delete them.

118567 The *ls* utility is also an exception to this rule. For compatibility with historical systems,
 118568 when the **-R** option is not specified, the *ls* utility follows symbolic links named as
 118569 arguments if the **-L** option is specified or if the **-F**, **-d**, or **-l** options are not specified. (If
 118570 the **-L** option is specified, *ls* always follows symbolic links; it is the only utility where the
 118571 **-L** option affects its behavior even though a tree walk is not being performed.)

118572 All other standard utilities, when not traversing a file hierarchy, always follow symbolic links
 118573 named as arguments.

118574 Historical practice is that the **-h** option is specified if standard utilities are to act upon symbolic
 118575 links instead of upon their targets. Examples of commands that have historically had a **-h** option
 118576 for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.

118577 *Third Domain*

118578 The third domain is symbolic links, referencing files not of type directory, specified to utilities
 118579 that are performing a traversal of a file hierarchy. (This includes symbolic links specified as
 118580 command line pathname arguments or encountered during the traversal.)

118581 The intention of the Shell and Utilities volume of POSIX.1-2017 is that the operation that the
 118582 utility is performing is applied to the symbolic link itself, if that operation is applicable to
 118583 symbolic links. If the operation is not applicable to symbolic links, the symbolic link should be
 118584 ignored. Specifically, by default, no change should be made to the file referenced by the symbolic
 118585 link.

118586 *Fourth Domain*

118587 The fourth domain is symbolic links referencing files of type directory, specified to utilities that
 118588 are performing a traversal of a file hierarchy. (This includes symbolic links specified as
 118589 command line pathname arguments or encountered during the traversal.)

118590 Most standard utilities do not, by default, indirect into the file hierarchy referenced by the
 118591 symbolic link. (The Shell and Utilities volume of POSIX.1-2017 uses the informal term “physical
 118592 walk” to describe this case. The case where the utility does indirect through the symbolic link is
 118593 termed a “logical walk”.)

118594 There are three reasons for the default to be a physical walk:

- 118595 1. With very few exceptions, a physical walk has been the historical default on UNIX
 118596 systems supporting symbolic links. Because some utilities (that is, *rm*) must default to a
 118597 physical walk, regardless, changing historical practice in this regard would be confusing
 118598 to users and needlessly incompatible.
- 118599 2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*,
 118600 *mode*), defaulting to a logical traversal would require the addition of a new option to the
 118601 commands to modify the attributes of the link itself. This is painful and more complex
 118602 than the alternatives.
- 118603 3. There is a security issue with defaulting to a logical walk. Historically, the command
 118604 *chown -R user file* has been safe for the superuser because *setuid* and *setgid* bits were lost
 118605 when the ownership of the file was changed. If the walk were logical, changing
 118606 ownership would no longer be safe because a user might have inserted a symbolic link
 118607 pointing to any file in the tree. Again, this would necessitate the addition of an option to
 118608 the commands doing hierarchy traversal to not indirect through the symbolic links, and
 118609 historical scripts doing recursive walks would instantly become security problems. While
 118610 this is mostly an issue for system administrators, it is preferable to not have different
 118611 defaults for different classes of users.

118612 However, the standard developers agreed to leave it unspecified to achieve consensus.

118613 As consistently as possible, users may cause standard utilities performing a file hierarchy
 118614 traversal to follow any symbolic links named on the command line, regardless of the type of file
 118615 they reference, by specifying the **-H** (for half logical) option. This option is intended to make the
 118616 command line name space look like the logical name space.

118617 As consistently as possible, users may cause standard utilities performing a file hierarchy
 118618 traversal to follow any symbolic links named on the command line as well as any symbolic links
 118619 encountered during the traversal, regardless of the type of file they reference, by specifying the
 118620 **-L** (for logical) option. This option is intended to make the entire name space look like the
 118621 logical name space.

118622 For consistency, implementors are encouraged to use the **-P** (for “physical”) flag to specify the
 118623 physical walk in utilities that do logical walks by default for whatever reason.

118624 When one or more of the **-H**, **-L**, and **-P** flags can be specified, the last one specified determines
 118625 the behavior of the utility. This permits users to alias commands so that the default behavior is a
 118626 logical walk and then override that behavior on the command line.

118627 *Exceptions in the Third and Fourth Domains*

118628 The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links
 118629 and does not support the **-H**, **-L**, or **-P** options. Some historical versions of *ls* always followed
 118630 symbolic links given on the command line whether the **-L** option was specified or not.
 118631 Historical versions of *ls* did not support the **-H** option. In POSIX.1-2017, unless one of the **-H** or
 118632 **-L** options is specified, the *ls* utility only follows symbolic links to directories that are given as
 118633 operands. The *ls* utility does not support the **-P** option.

118634 The Shell and Utilities volume of POSIX.1-2017 requires that the standard utilities *ls*, *find*, and
 118635 *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a
 118636 symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is
 118637 corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often
 118638 used in system administration and security applications, they should attempt to recover and
 118639 continue as best as they can. The *pax* utility should terminate because the archive it was creating
 118640 is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.
 118641 Implementations are strongly encouraged to detect infinite loops in all utilities.

118642 Historical practice is shown in [Table A-1](#). The heading **SVID3** stands for the Third Edition of the
118643 System V Interface Definition.

118644 Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,
118645 *pwd* reported the physical path, and in others, the logical path. Implementations of the shell
118646 corresponding to POSIX.1-2017 must report the logical path by default.

118647 The *cd* command is required, by default, to treat the filename dot-dot logically. Implementors are
118648 required to support the **-P** flag in *cd* so that users can have their current environment handled
118649 physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic link, not
118650 the target. Symbolic links in 4.4 BSD did not have *owner*, *group*, *mode*, or other standard UNIX
118651 system file attributes.

118652 **Table A-1** Historical Practice for Symbolic Links

Utility	SVID3	4.3 BSD	4.4 BSD	POSIX	Comments
118653 <i>cd</i>				-L	Treat ". ." logically.
118654 <i>cd</i>				-P	Treat ". ." physically.
118655 <i>chgrp</i>			-H	-H	Follow command line symlinks.
118656 <i>chgrp</i>			-h	-L	Follow symlinks.
118657 <i>chgrp</i>	-h			-h	Affect the symlink.
118658 <i>chmod</i>					Affect the symlink.
118659 <i>chmod</i>			-H		Follow command line symlinks.
118660 <i>chmod</i>			-h		Follow symlinks.
118661 <i>chown</i>			-H	-H	Follow command line symlinks.
118662 <i>chown</i>			-h	-L	Follow symlinks.
118663 <i>chown</i>	-h			-h	Affect the symlink.
118664 <i>cp</i>			-H	-H	Follow command line symlinks.
118665 <i>cp</i>			-h	-L	Follow symlinks.
118666 <i>cpio</i>	-L		-L		Follow symlinks.
118667 <i>du</i>			-H	-H	Follow command line symlinks.
118668 <i>du</i>			-h	-L	Follow symlinks.
118669 <i>file</i>	-h			-h	Affect the symlink.
118670 <i>find</i>			-H	-H	Follow command line symlinks.
118671 <i>find</i>			-h	-L	Follow symlinks.
118672 <i>find</i>	-follow		-follow		Follow symlinks.
118673 <i>ln</i>	-s	-s	-s	-s	Create a symbolic link.
118674 <i>ls</i>	-L	-L	-L	-L	Follow symlinks.
118675 <i>ls</i>				-H	Follow command line symlinks.
118676 <i>mv</i>					Operates on the symlink.
118677 <i>pax</i>			-H	-H	Follow command line symlinks.
118678 <i>pax</i>			-h	-L	Follow symlinks.
118679 <i>pwd</i>				-L	Printed path may contain symlinks.
118680 <i>pwd</i>				-P	Printed path will not contain symlinks.
118681 <i>rm</i>					Operates on the symlink.
118682 <i>tar</i>			-H		Follow command line symlinks.
118683 <i>tar</i>		-h	-h		Follow symlinks.
118684 <i>test</i>	-h		-h	-h	Affect the symlink.

118686 Synchronously-Generated Signal

118687 Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE,
118688 SIGPIPE, and SIGSEGV.

118689 Any signal sent via the *raise()* function or a *kill()* function targeting the current process is also
118690 considered synchronous.

118691 System Call*

118692 The distinction between a “system call” and a “library routine” is an implementation detail that
118693 may differ between implementations and has thus been excluded from POSIX.1.

118694 See “Interface, Not Implementation” in the Preface.

118695 System Console

118696 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/7 is applied, changing from “An
118697 implementation-defined device” to “A device”.

118698 System Databases

118699 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/9 is applied, rewording the definition to
118700 reference the existing definitions for “group database” and “user database”.

118701 System Process

118702 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/8 is applied, rewording the definition to
118703 remove the requirement for an implementation to define the object.

118704 System Reboot

118705 A “system reboot” is an event initiated by an unspecified circumstance that causes all processes
118706 (other than special system processes) to be terminated in an implementation-defined manner,
118707 after which any changes to the state and contents of files created or written to by a Conforming
118708 POSIX.1 Application prior to the event are implementation-defined.

118709 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/10 is applied, changing “An
118710 implementation-defined sequence of events” to “An unspecified sequence of events”.

118711 Synchronized I/O Data (and File) Integrity Completion

118712 These terms specify that for synchronized read operations, pending writes must be successfully
118713 completed before the read operation can complete. This is motivated by two circumstances.
118714 Firstly, when synchronizing processes can access the same file, but not share common buffers
118715 (such as for a remote file system), this requirement permits the reading process to guarantee that
118716 it can read data written remotely. Secondly, having data written synchronously is insufficient to
118717 guarantee the order with respect to a subsequent write by a reading process, and thus this extra
118718 read semantic is necessary.

118719 Text File

118720 The term “text file” does not prevent the inclusion of control or other non-printable characters
118721 (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either
118722 able to process the special characters or they explicitly describe their limitations within their
118723 individual descriptions. The definition of “text file” has caused controversy. The only difference
118724 between text and binary files is that text files have lines of less than {LINE_MAX} bytes, with no
118725 NUL characters, each terminated by a <newline>. The definition allows a file with a single
118726 <newline>, or a totally empty file, to be called a text file. If a file ends with an incomplete line it
118727 is not strictly a text file by this definition. The <newline> referred to in POSIX.1-2017 is not some
118728 generic line separator, but a single character; files created on systems where they use multiple
118729 characters for ends of lines are not portable to all conforming systems without some translation
118730 process unspecified by POSIX.1-2017.

118731 Thread

118732 POSIX.1-2017 defines a thread to be a flow of control within a process. Each thread has a
118733 minimal amount of private state; most of the state associated with a process is shared among all
118734 of the threads in the process. While most multi-thread extensions to POSIX have taken this
118735 approach, others have made different decisions.

118736 **Note:** The choice to put threads within a process does not constrain implementations to implement
118737 threads in that manner. However, all functions have to behave as though threads share the
118738 indicated state information with the process from which they were created.

118739 Threads need to share resources in order to cooperate. Memory has to be widely shared between
118740 threads in order for the threads to cooperate at a fine level of granularity. Threads keep data
118741 structures and the locks protecting those data structures in shared memory. For a data structure
118742 to be usefully shared between threads, such structures should not refer to any data that can only
118743 be interpreted meaningfully by a single thread. Thus, any system resources that might be
118744 referred to in data structures need to be shared between all threads. File descriptors, pathnames,
118745 and pointers to stack variables are all things that programmers want to share between their
118746 threads. Thus, the file descriptor table, the root directory, the current working directory, and the
118747 address space have to be shared.

118748 Library implementations are possible as long as the effective behavior is as if system services
118749 invoked by one thread do not suspend other threads. This may be difficult for some library
118750 implementations on systems that do not provide asynchronous facilities.

118751 See [Section B.2.9](#) (on page 3633) for additional rationale.

118752 Thread ID

118753 See [Section B.2.9.2](#) (on page 3650) for additional rationale.

118754 Thread-Safe Function

118755 All functions required by POSIX.1-2017 need to be thread-safe; see [Section A.4.18](#) (on page 3520)
118756 and [Section B.2.9.1](#) (on page 3647) for additional rationale.

118757 **User Database**

118758 There are no references in POSIX.1-2017 to a “passwd file” or a “group file”, and there is no
 118759 requirement that the *group* or *passwd* databases be kept in files containing editable text. Many
 118760 large timesharing systems use *passwd* databases that are hashed for speed. Certain security
 118761 classifications prohibit certain information in the *passwd* database from being publicly readable.

118762 The term “encoded” is used instead of “encrypted” in order to avoid the implementation
 118763 connotations (such as reversibility or use of a particular algorithm) of the latter term.

118764 The *getgrent()*, *setgrent()*, *endgrent()*, *getpwent()*, *setpwent()*, and *endpwent()* functions are not
 118765 included as part of the base standard because they provide a linear database search capability
 118766 that is not generally useful (the *getpwuid()*, *getpwnam()*, *getgrgid()*, and *getgrnam()* functions are
 118767 provided for keyed lookup) and because in certain distributed systems, especially those with
 118768 different authentication domains, it may not be possible or desirable to provide an application
 118769 with the ability to browse the system databases indiscriminately. They are provided on XSI-
 118770 conformant systems due to their historical usage by many existing applications.

118771 A change from historical implementations is that the structures used by these functions have
 118772 fields of the types **gid_t** and **uid_t**, which are required to be defined in the **<sys/types.h>** header.
 118773 POSIX.1-2017 requires implementations to ensure that these types are defined by inclusion of
 118774 **<grp.h>** and **<pwd.h>**, respectively, without imposing any name space pollution or errors from
 118775 redefinition of types.

118776 POSIX.1-2017 is silent about the content of the strings containing user or group names. These
 118777 could be digit strings. POSIX.1-2017 is also silent as to whether such digit strings bear any
 118778 relationship to the corresponding (numeric) user or group ID.

118779 *Database Access*

118780 The thread-safe versions of the user and group database access functions return values in user-
 118781 supplied buffers instead of possibly using static data areas that may be overwritten by each call.

118782 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/11 is applied, removing the words “of
 118783 implementation-defined format”.

118784 **User ID**

118785 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0016 [511] is applied.

118786 **User Name**

118787 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0017 [584] is applied.

118788 **Virtual Processor***

118789 The term “virtual processor” was chosen as a neutral term describing all kernel-level
 118790 schedulable entities, such as processes, Mach tasks, or lightweight processes. Implementing
 118791 threads using multiple processes as virtual processors, or implementing multiplexed threads
 118792 above a virtual processor layer, should be possible, provided some mechanism has also been
 118793 implemented for sharing state between processes or virtual processors. Many systems may also
 118794 wish to provide implementations of threads on systems providing “shared processes” or
 118795 “variable-weight processes”. It was felt that exposing such implementation details would
 118796 severely limit the type of systems upon which the threads interface could be supported and
 118797 prevent certain types of valid implementations. It was also determined that a virtual processor
 118798 interface was out of the scope of the Rationale (Informative) volume of POSIX.1-2017.

- 118799 **XSI**
- 118800 This is included to allow POSIX.1-2017 to be adopted as an IEEE standard and an Open Group
118801 Standard, serving both the POSIX and the Single UNIX Specification in a core set of volumes.
- 118802 The term “XSI” has been used for 10 years in connection with the XPG series and the first and
118803 second versions of the base volumes of the Single UNIX Specification. The XSI margin code was
118804 introduced to denote the extended or more restrictive semantics beyond POSIX that are
118805 applicable to UNIX systems.
- 118806 **Zombie Process**
- 118807 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0018 [690] is applied.
-
- 118808 **A.4 General Concepts**
- 118809 The general concepts are similar in nature to the definitions section, with the exception that a
118810 term defined in general concepts can contain normative requirements.
-
- 118811 **A.4.1 Concurrent Execution**
- 118812 There is no additional rationale provided for this section.
-
- 118813 **A.4.2 Default Initialization**
- 118814 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0019 [934] is applied.
-
- 118815 **A.4.3 Directory Protection**
- 118816 There is no additional rationale provided for this section.
-
- 118817 **A.4.4 Extended Security Controls**
- 118818 Allowing an implementation to define extended security controls enables the use of
118819 POSIX.1-2017 in environments that require different or more rigorous security than that
118820 provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions.
118821 The semantics of these areas have been defined to permit extensions with reasonable, but not
118822 exact, compatibility with all existing practices. For example, the elimination of the superuser
118823 definition precludes identifying a process as privileged or not by virtue of its effective user ID.
-
- 118824 **A.4.5 File Access Permissions**
- 118825 A process should not try to anticipate the result of an attempt to access data by *a priori* use of
118826 these rules. Rather, it should make the attempt to access data and examine the return value (and
118827 possibly *errno* as well), or use *access()*. An implementation may include other security
118828 mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of
118829 those additional mechanisms, even though it would succeed according to the rules given in this
118830 section. (For example, the user’s security level might be lower than that of the object of the
118831 access attempt.) The supplementary group IDs provide another reason for a process to not

118832 attempt to anticipate the result of an access attempt.

118833 Since the current standard does not specify a method for opening a directory for searching, it is
 118834 unspecified whether search permission on the *fd* argument to *openat()* and related functions is
 118835 based on whether the directory was opened with search mode or on the current permissions
 118836 allowed by the directory at the time a search is performed. When there is existing practice that
 118837 supports opening directories for searching, it is expected that these functions will be modified to
 118838 specify that the search permissions will be granted based on the file access modes of the
 118839 directory's file descriptor identified by *fd*, and not on the mode of the directory at the time the
 118840 directory is searched.

118841 A.4.6 File Hierarchy

118842 Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such
 118843 for three reasons:

- 118844 1. Links may join branches.
- 118845 2. In some network implementations, there may be no single absolute root directory; see
 118846 *pathname resolution*.
- 118847 3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

118848 A.4.7 Filenames

118849 Historically, certain filenames and pathnames have been reserved. This list includes **core**,
 118850 **/etc/passwd**, and so on. Conforming applications should avoid these.

118851 Most historical implementations prohibit case folding in filenames; that is, treating uppercase
 118852 and lowercase alphabetic characters as identical. However, some consider case folding desirable:

118853 For user convenience

118854 For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular
 118855 operating systems

118856 Variants, such as maintaining case distinctions in filenames, but ignoring them in comparisons,
 118857 have been suggested. Methods of allowing escaped characters of the case opposite the default
 118858 have been proposed.

118859 Many reasons have been expressed for not allowing case folding, including:

118860 No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is
 118861 more convenient for users.

118862 Making case-insensitivity a POSIX.1 implementation option would be worse than either
 118863 having it or not having it, because:

118864 † ~~OM~~ confusion would be caused among users.

118865 † ~~pl~~ application developers would have to account for both cases in their code.

118866 † ~~OS~~ POSIX.1 implementors would still have other problems with native file systems, such
 118867 as short or otherwise constrained filenames or pathnames, and the lack of
 118868 hierarchical directory structure.

118869 Case folding is not easily defined in many European languages, both because many of
 118870 them use characters outside the US ASCII alphabetic set, and because:

- 118871 In Spanish, the digraph "ll" is considered to be a single letter, the capitalized form
118872 of which may be either "Ll" or "LL", depending on context.
- 118873 In French, the capitalized form of a letter with an accent may or may not retain the
118874 accent, depending on the country in which it is written.
- 118875 In German, the sharp ess may be represented as a single character resembling a
118876 Greek beta (β) in lowercase, but as the digraph "SS" in uppercase.
- 118877 In Greek, there are several lowercase forms of some letters; the one to use depends on
118878 its position in the word. Arabic has similar rules.
- 118879 Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish
118880 case and are sometimes encoded in character sets that use more than one byte per
118881 character.
- 118882 Multiple character codes may be used on the same machine simultaneously. There are
118883 several ISO character sets for European alphabets. In Japan, several Japanese character
118884 codes are commonly used together, sometimes even in filenames; this is evidently also the
118885 case in China. To handle case insensitivity, the kernel would have to at least be able to
118886 distinguish for which character sets the concept made sense.
- 118887 The file system implementation historically deals only with bytes, not with characters.
118888 Limitations on valid encodings ensure that the byte sequences for the <slash> character,
118889 <period> character, and <NUL> character will not be confused with any other character in
118890 any locale. However, there exist common single-shift encodings where other single-byte
118891 characters from the portable filename character set can also occur as a subset of a multi-
118892 byte character, making case folding of portable filename bytes dependent on the context of
118893 whether a shift-state is active.
- 118894 The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.
118895 Mandating case-insensitivity would make all historical implementations non-standard.
- 118896 Not only the interface, but also application programs would need to change, counter to the
118897 purpose of having minimal changes to existing application code.
- 118898 At least one of the original developers of the UNIX system has expressed objection in the
118899 strongest terms to either requiring case-insensitivity or making it an option, mostly on the
118900 basis that POSIX.1 should not hinder portability of application programs across related
118901 implementations in order to allow compatibility with unrelated operating systems.
- 118902 Two proposals were entertained regarding case folding in filenames:
- 118903 1. Remove all wording that previously permitted case folding.
- 118904 Rationale Case folding is inconsistent with the portable filename character set and
118905 filename definitions (all bytes except <slash> and null). No known
118906 implementations allowing all bytes except <slash> and null also do case
118907 folding.
- 118908 2. Change "though this practice is not recommended:" to "although this practice is strongly
118909 discouraged."
- 118910 Rationale If case folding must be included in POSIX.1, the wording should be stronger
118911 to discourage the practice.
- 118912 The consensus selected the first proposal. Otherwise, a conforming application would have to
118913 assume that case folding would occur when it was not wanted, but that it would not occur when
118914 it was wanted.

118915 A.4.8 Filename Portability

118916 Filenames should be constructed from the portable filename character set because the use of
118917 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a
118918 <colon> (' : ') in a pathname could cause ambiguity if that pathname were included in a *PATH*
118919 definition.)

118920 The constraint on use of the <hyphen-minus> character as the first character of a portable
118921 filename is a constraint on application behavior and not on implementations, since applications
118922 might not work as expected when such a filename is passed as a command line argument.

118923 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0020 [584] is applied.

118924 A.4.9 File Times Update

118925 This section reflects the actions of historical implementations. The times are not updated
118926 immediately, but are only marked for update by the functions. An implementation may update
118927 these times immediately.

118928 The accuracy of the time update values is intentionally left unspecified so that systems can
118929 control the bandwidth of a possible covert channel.

118930 The wording was carefully chosen to make it clear that there is no requirement that the
118931 conformance document contain information that might incidentally affect file timestamps. Any
118932 function that performs pathname resolution might update several last data access timestamps.
118933 Functions such as *getpwnam()* and *getgrnam()* might update the last data access timestamp of
118934 some specific file or files. It is intended that these are not required to be documented in the
118935 conformance document, but they should appear in the system documentation.

118936 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0021 [626] is applied.

118937 A.4.10 Host and Network Byte Order

118938 There is no additional rationale provided for this section.

118939 A.4.11 Measurement of Execution Time

118940 The methods used to measure the execution time of processes and threads, and the precision of
118941 these measurements, may vary considerably depending on the software architecture of the
118942 implementation, and on the underlying hardware. Implementations can also make tradeoffs
118943 between the scheduling overhead and the precision of the execution time measurements.
118944 POSIX.1-2017 does not impose any requirement on the accuracy of the execution time; it instead
118945 specifies that the measurement mechanism and its precision are implementation-defined.

118946 **A.4.12 Memory Synchronization**

118947 In older multi-processors, access to memory by the processors was strictly multiplexed. This
 118948 meant that a processor executing program code interrogates or modifies memory in the order
 118949 specified by the code and that all the memory operation of all the processors in the system
 118950 appear to happen in some global order, though the operation histories of different processors are
 118951 interleaved arbitrarily. The memory operations of such machines are said to be sequentially
 118952 consistent. In this environment, threads can synchronize using ordinary memory operations. For
 118953 example, a producer thread and a consumer thread can synchronize access to a circular data
 118954 buffer as follows:

```
118955 int rdptr = 0;
118956 int wrptr = 0;
118957 data_t buf[BUFSIZE];

118958 Thread 1:
118959     while (work_to_do) {
118960         int next;

118961         buf[wrptr] = produce();
118962         next = (wrptr + 1) % BUFSIZE;
118963         while (rdptr == next)
118964             ;
118965         wrptr = next;
118966     }

118967 Thread 2:
118968     while (work_to_do) {
118969         while (rdptr == wrptr)
118970             ;
118971         consume(buf[rdptr]);
118972         rdptr = (rdptr + 1) % BUFSIZE;
118973     }
```

118974 In modern multi-processors, these conditions are relaxed to achieve greater performance. If one
 118975 processor stores values in location A and then location B, then other processors loading data
 118976 from location B and then location A may see the new value of B but the old value of A. The
 118977 memory operations of such machines are said to be weakly ordered. On these machines, the
 118978 circular buffer technique shown in the example will fail because the consumer may see the new
 118979 value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can
 118980 only be achieved through the use of special instructions that enforce an order on memory
 118981 operations. Most high-level language compilers only generate ordinary memory operations to
 118982 take advantage of the increased performance. They usually cannot determine when memory
 118983 operation order is important and generate the special ordering instructions. Instead, they rely on
 118984 the programmer to use synchronization primitives correctly to ensure that modifications to a
 118985 location in memory are ordered with respect to modifications and/or access to the same location
 118986 in other threads. Access to read-only data need not be synchronized. The resulting program is
 118987 said to be data race-free.

118988 Synchronization is still important even when accessing a single primitive variable (for example,
 118989 an integer). On machines where the integer may not be aligned to the bus data width or be
 118990 larger than the data width, a single memory load may require multiple memory cycles. This
 118991 means that it may be possible for some parts of the integer to have an old value while other
 118992 parts have a newer value. On some processor architectures this cannot happen, but portable
 118993 programs cannot rely on this.

118994 In summary, a portable multi-threaded program, or a multi-process program that shares

118995 writable memory between processes, has to use the synchronization primitives to synchronize
 118996 data access. It cannot rely on modifications to memory being observed by other threads in the
 118997 order written in the application or even on modification of a single variable being seen
 118998 atomically.

118999 Conforming applications may only use the functions listed to synchronize threads of control
 119000 with respect to memory access. There are many other candidates for functions that might also be
 119001 used. Examples are: signal sending and reception, or pipe writing and reading. In general, any
 119002 function that allows one thread of control to wait for an action caused by another thread of
 119003 control is a candidate. POSIX.1-2017 does not require these additional functions to synchronize
 119004 memory access since this would imply the following:

119005 All these functions would have to be recognized by advanced compilation systems so that
 119006 memory operations and calls to these functions are not reordered by optimization.

119007 All these functions would potentially have to have memory synchronization instructions
 119008 added, depending on the particular machine.

119009 The additional functions complicate the model of how memory is synchronized and make
 119010 automatic data race detection techniques impractical.

119011 Formal definitions of the memory model were rejected as unreadable by the vast majority of
 119012 programmers. In addition, most of the formal work in the literature has concentrated on the
 119013 memory as provided by the hardware as opposed to the application programmer through the
 119014 compiler and runtime system. It was believed that a simple statement intuitive to most
 119015 programmers would be most effective. POSIX.1-2017 defines functions that can be used to
 119016 synchronize access to memory, but it leaves open exactly how one relates those functions to the
 119017 semantics of each function as specified elsewhere in POSIX.1-2017. POSIX.1-2017 also does not
 119018 make a formal specification of the partial ordering in time that the functions can impose, as that
 119019 is implied in the description of the semantics of each function. It simply states that the
 119020 programmer has to ensure that modifications do not occur “simultaneously” with other access
 119021 to a memory location.

119022 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/4 is applied, adding a new paragraph
 119023 beneath the table of functions: “The *pthread_once()* function shall synchronize memory for the
 119024 first call in each thread for a given **pthread_once_t** object.”.

119025 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0022 [863] is applied.

119026 **A.4.13 Pathname Resolution**

119027 It is necessary to differentiate between the definition of pathname and the concept of pathname
 119028 resolution with respect to the handling of trailing <slash> characters. By specifying the behavior
 119029 here, it is not possible to provide an implementation that is conforming but extends all interfaces
 119030 that handle pathnames to also handle strings that are not legal pathnames (because they have
 119031 trailing <slash> characters).

119032 Pathnames that end with one or more trailing <slash> characters must refer to directory paths.
 119033 Earlier versions of this standard were not specific about the distinction between trailing <slash>
 119034 characters on files and directories, and both were permitted.

119035 Two types of implementation have been prevalent; those that ignored trailing <slash> characters
 119036 on all pathnames regardless, and those that permitted them only on existing directories.

119037 An earlier version of this standard required that a pathname with a trailing <slash> character be
 119038 treated as if it had a trailing " / ." everywhere. This specification was ambiguous. In situations
 119039 where the intent was that the application wanted to require the implementation to accept the

119040 pathname only if it named a directory (existing or to be created as a result of the call performing
 119041 pathname resolution), literally adding a '.' after the trailing <slash> could be interpreted to
 119042 require use of that pathname to fail. Some of the uses that created ambiguous requirements
 119043 included `mkdir("newdir/")` and `rmdir("existing-dir/")`. POSIX.1-2017 requires that a pathname
 119044 with a trailing <slash> be rejected unless it refers to a file that is a directory or to a file that is to
 119045 be created as a directory. The `rename()` function and the `mv` utility further specify that a trailing
 119046 <slash> cannot be used on a pathname naming a file that does not exist when used as the last
 119047 argument to `rename()` or `renameat()`, or as the last operand to `mv`.

119048 Note that this change does not break any conforming applications; since there were two different
 119049 types of implementation, no application could have portably depended on either behavior. This
 119050 change does however require some implementations to be altered to remain compliant.
 119051 Substantial discussion over a three-year period has shown that the benefits to application
 119052 developers outweighs the disadvantages for some vendors.

119053 On a historical note, some early applications automatically appended a '/' to every path.
 119054 Rather than fix the applications, the system implementation was modified to accept this
 119055 behavior by ignoring any trailing <slash>.

119056 Each directory has exactly one parent directory which is represented by the name **dot-dot** in the
 119057 first directory. No other directory, regardless of linkages established by symbolic links, is
 119058 considered the parent directory by POSIX.1-2017.

119059 There are two general categories of interfaces involving pathname resolution: those that follow
 119060 the symbolic link, and those that do not. There are several exceptions to this rule; for example,
 119061 `open(path,O_CREAT|O_EXCL)` will fail when `path` names a symbolic link. However, in all other
 119062 situations, the `open()` function will follow the link.

119063 What the filename **dot-dot** refers to relative to the root directory is implementation-defined. In
 119064 Version 7 it refers to the root directory itself; this is the behavior mentioned in POSIX.1-2017. In
 119065 some networked systems the construction `./hostname/` is used to refer to the root directory of
 119066 another host, and POSIX.1 permits this behavior.

119067 Other networked systems use the construct `//hostname` for the same purpose; that is, a double
 119068 initial <slash> is used. There is a potential problem with existing applications that create full
 119069 pathnames by taking a trunk and a relative pathname and making them into a single string
 119070 separated by '/', because they can accidentally create networked pathnames when the trunk is
 119071 '/'. This practice is not prohibited because such applications can be made to conform by
 119072 simply changing to use "//" as a separator instead of '/':

119073 If the trunk is '/', the full pathname will begin with "///" (the initial '/' and the
 119074 separator "//"). This is the same as '/', which is what is desired. (This is the general
 119075 case of making a relative pathname into an absolute one by prefixing with "///" instead
 119076 of '/'.)

119077 If the trunk is "/A", the result is "/A//..."; since non-leading sequences of two or more
 119078 <slash> characters are treated as a single <slash>, this is equivalent to the desired
 119079 "/A/...".

119080 If the trunk is "//A", the implementation-defined semantics will apply. (The multiple
 119081 <slash> rule would apply.)

119082 Application developers should avoid generating pathnames that start with "//".
 119083 Implementations are strongly encouraged to avoid using this special interpretation since a
 119084 number of applications currently do not follow this practice and may inadvertently generate
 119085 "//...".

119086 The term "root directory" is only defined in POSIX.1 relative to the process. In some

119087 implementations, there may be no absolute root directory. The initialization of the root directory
119088 of a process is implementation-defined.

119089 When the standard says: “Pathname resolution for a given pathname shall yield the same results
119090 when used by any interface in POSIX.1-2017 as long as there are no changes to any files
119091 evaluated during pathname resolution for the given pathname between resolutions”, this
119092 applies to absolute pathnames or to relative pathnames from the same current working
119093 directory. Using the same relative pathname from two different working directories may yield
119094 different results.

119095 Earlier versions of this standard were unclear as to whether a pathname was required to be a
119096 character string or just a string. This standard is now clear that filenames are just strings, and
119097 that pathname processing is locale-independent.

119098 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0023 [541,649,825] and
119099 XBD/TC2-2008/0024 [825] are applied.

119100 **A.4.14 Process ID Reuse**

119101 There is no additional rationale provided for this section.

119102 **A.4.15 Scheduling Policy**

119103 There is no additional rationale provided for this section.

119104 **A.4.16 Seconds Since the Epoch**

119105 Coordinated Universal Time (UTC) includes leap seconds. However, in POSIX time (seconds
119106 since the Epoch), leap seconds are ignored (not applied) to provide an easy and compatible
119107 method of computing time differences. Broken-down POSIX time is therefore not necessarily
119108 UTC, despite its appearance.

119109 As of December 2007, 23 leap seconds had been added to UTC since the Epoch, 1 January, 1970.
119110 Historically, one leap second is added every 15 months on average, so this offset can be expected
119111 to grow with time.

119112 Most systems’ notion of “time” is that of a continuously increasing value, so this value should
119113 increase even during leap seconds. However, not only do most systems not keep track of leap
119114 seconds, but most systems are probably not synchronized to any standard time reference.
119115 Therefore, it is inappropriate to require that a time represented as seconds since the Epoch
119116 precisely represent the number of seconds between the referenced time and the Epoch.

119117 It is sufficient to require that applications be allowed to treat this time as if it represented the
119118 number of seconds between the referenced time and the Epoch. It is the responsibility of the
119119 vendor of the system, and the administrator of the system, to ensure that this value represents
119120 the number of seconds between the referenced time and the Epoch as closely as necessary for the
119121 application being run on that system.

119122 It is important that the interpretation of time names and seconds since the Epoch values be
119123 consistent across conforming systems; that is, it is important that all conforming systems
119124 interpret “536 457 599 seconds since the Epoch” as 59 seconds, 59 minutes, 23 hours 31 December
119125 1986, regardless of the accuracy of the system’s idea of the current time. The expression is given
119126 to ensure a consistent interpretation, not to attempt to specify the calendar. The relationship
119127 between *tm_yday* and the day of week, day of month, and month is in accordance with the

119128 Gregorian calendar, and so is not specified in POSIX.1.

119129 Consistent interpretation of seconds since the Epoch can be critical to certain types of distributed
119130 applications that rely on such timestamps to synchronize events. The accrual of leap seconds in a
119131 time standard is not predictable. The number of leap seconds since the Epoch will likely
119132 increase. POSIX.1 is more concerned about the synchronization of time between applications of
119133 astronomically short duration.

119134 Note that *tm_yday* is zero-based, not one-based, so the day number in the example above is 364.
119135 Note also that the division is an integer division (discarding remainder) as in the C language.

119136 Note also that the meaning of *gmtime()*, *localtime()*, and *mktime()* is specified in terms of this
119137 expression. However, the ISO C standard computes *tm_yday* from *tm_mday*, *tm_mon*, and *tm_year*
119138 in *mktime()*. Because it is stated as a (bidirectional) relationship, not a function, and because the
119139 conversion between month-day-year and day-of-year dates is presumed well known and is also
119140 a relationship, this is not a problem.

119141 Implementations that implement **time_t** as a signed 32-bit integer will overflow in 2038. This
119142 standard requires that **time_t** be an integer type with implementation-defined size, but does not
119143 mandate a particular size. The requirement that **time_t** be integral is an additional constraint
119144 beyond the ISO C standard, which allows a real-floating **time_t**. Implementation practice has
119145 shown that much existing code is unprepared to deal with a floating-point **time_t**, and that use
119146 of **struct timespec** is a more uniform way to provide sub-second time manipulation within
119147 applications.

119148 See also [Epoch](#) (on page 3492).

119149 The topic of whether seconds since the Epoch should account for leap seconds has been debated
119150 on a number of occasions, and each time consensus was reached (with acknowledged dissent
119151 each time) that the majority of users are best served by treating all days identically. (That is, the
119152 majority of applications were judged to assume a single length—as measured in seconds since
119153 the Epoch—for all days. Thus, leap seconds are not applied to seconds since the Epoch.) Those
119154 applications which do care about leap seconds can determine how to handle them in whatever
119155 way those applications feel is best. This was particularly emphasized because there was
119156 disagreement about what the best way of handling leap seconds might be. It is a practical
119157 impossibility to mandate that a conforming implementation must have a fixed relationship to
119158 any particular official clock (consider isolated systems, or systems performing “reruns” by
119159 setting the clock to some arbitrary time).

119160 Note that as a practical consequence of this, the length of a second as measured by some external
119161 standard is not specified. This unspecified second is nominally equal to an International System
119162 (SI) second in duration. Applications must be matched to a system that provides the particular
119163 handling of external time in the way required by the application.

119164 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/12 is applied, making an editorial
119165 correction to the paragraph commencing “How any changes to the value of seconds ...”.

119166 A.4.17 Semaphore

119167 There is no additional rationale provided for this section.

119168 **A.4.18 Thread-Safety**

119169 Where the interface of a function required by POSIX.1-2017 precludes thread-safety, an alternate
 119170 thread-safe form is provided. The names of these thread-safe forms are the same as the non-
 119171 thread-safe forms with the addition of the suffix ``_r''. The suffix ``_r'' is historical, where the
 119172 'r' stood for ``reentrant''.

119173 In some cases, thread-safety is provided by restricting the arguments to an existing function.

119174 See also [Section B.2.9.1](#) (on page 3647).

119175 **A.4.19 Tracing**

119176 Refer to [Section B.2.11](#) (on page 3664).

119177 **A.4.20 Treatment of Error Conditions for Mathematical Functions**

119178 It is intended that undeserved underflow and inexact floating-point exceptions are raised only if
 119179 avoiding them would be too costly.

119180 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0025 [543] is applied.

119181 **A.4.21 Treatment of NaN Arguments for Mathematical Functions**

119182 There is no additional rationale provided for this section.

119183 **A.4.22 Utility**

119184 There is no additional rationale provided for this section.

119185 **A.4.23 Variable Assignment**

119186 There is no additional rationale provided for this section.

119187 **A.5 File Format Notation**

119188 The notation for spaces allows some flexibility for application output. Note that an empty
 119189 character position in *format* represents one or more <blank> characters on the output (not *white*
 119190 *space*, which can include <newline> characters). Therefore, another utility that reads that output
 119191 as its input must be prepared to parse the data using *scanf()*, *awk*, and so on. The 'Δ' character
 119192 is used when exactly one <space> is output.

119193 The treatment of integers and spaces is different from the *printf()* function in that they can be
 119194 surrounded with <blank> characters. This was done so that, given a format such as:

119195 `"%d\Δ", <foo>`

119196 the implementation could use a *printf()* call such as:

119197 `printf("%6d\Δ", foo);`

119198 and still conform. This notation is thus somewhat like *scanf()* in addition to *printf()*.

119199 The *printf()* function was chosen as a model because most of the standard developers were
 119200 familiar with it. One difference from the C function *printf()* is that the *l* and *h* conversion
 119201 specifier characters are not used. As expressed by the Shell and Utilities volume of
 119202 POSIX.1-2017, there is no differentiation between decimal values for type **int**, type **long**, or type
 119203 **short**. The conversion specifications *%d* or *%i* should be interpreted as an arbitrary length
 119204 sequence of digits. Also, no distinction is made between single precision and double precision
 119205 numbers (**float** or **double** in C). These are simply referred to as floating-point numbers.

119206 Many of the output descriptions in the Shell and Utilities volume of POSIX.1-2017 use the term
 119207 “line”, such as:

119208 “%s”, <input line>

119209 Since the definition of *line* includes the trailing <newline> already, there is no need to include a
 119210 ‘\n’ in the format; a double <newline> would otherwise result.

119211 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0026 [584] is applied.

119212 **A.6 Character Set**

119213 **A.6.1 Portable Character Set**

119214 The portable character set is listed in full so there is no dependency on the ISO/IEC 646:1991
 119215 standard (or historically ASCII) encoded character set, although the set is identical to the
 119216 characters defined in the International Reference version of the ISO/IEC 646:1991 standard.

119217 POSIX.1-2017 poses no requirement that multiple character sets or codesets be supported,
 119218 leaving this as a marketing differentiation for implementors. Although multiple charmap files
 119219 are supported, it is the responsibility of the implementation to provide the file(s); if only one is
 119220 provided, only that one will be accessible using the *localedef -f* option.

119221 The statement about invariance in codesets for the portable character set is worded to avoid
 119222 precluding implementations where multiple incompatible codesets are available (for instance,
 119223 ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if
 119224 they access portable characters that vary on the same implementation.

119225 Not all character sets need include the portable character set, but each locale must include it. For
 119226 example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201
 119227 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201
 119228 Katakana. Not all of these character sets include the portable characters, but at least one does
 119229 (JIS X 0201 Roman).

119230 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0027 [584,967] and
 119231 XBD/TC2-2008/0028 [745] are applied.

119232 A.6.2 Character Encoding

119233 Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and
119234 other countries, can be supported via the current charmap mechanism. With single-shift
119235 encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code,
119236 consisting of the portable character set (G0) and up to three additional character sets (G1, G2,
119237 G3), can be described using the current charmap mechanism; the encoding for each character in
119238 additional character sets G2 and G3 must then include their single-shift code. Other mechanisms
119239 to support locales based on encoding mechanisms such as locking shift are not addressed by this
119240 volume of POSIX.1-2017.

119241 The encodings for <slash> and <period> are required to be the same across all locales, in part
119242 because pathname resolution requires recognition of these bytes. It is a fortunate accident that all
119243 common shift-based encodings did not use either <slash> or <period> as a valid second byte in
119244 a multi-byte character.

119245 The encodings for <newline> and <carriage-return> are required to be the same across all
119246 locales since they are special to the general terminal interface and cannot be changed (see XBD
119247 [Section 11.1.9](#), on page 204).

119248 Earlier versions of this standard did not state the requirement that the POSIX locale contains 256
119249 single-byte characters. This was an oversight; the intention was always that the POSIX locale
119250 should have an 8-bit-clean single-byte encoding.

119251 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0029 [663,967] and
119252 XBD/TC2-2008/0030 [745] are applied.

119253 A.6.3 C Language Wide-Character Codes

119254 The standard does not specify how wide characters are encoded or provide a method for
119255 defining wide characters in a charmap. It specifies ways of translating between wide characters
119256 and multi-byte characters. The standard does not prevent an extension from providing a method
119257 to define wide characters.

119258 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/13 is applied, adding a statement that the
119259 standard has no means of defining a wide-character codeset.

119260 A.6.4 Character Set Description File

119261 IEEE PASC Interpretation 1003.2 #196 is applied, removing three lines of text dealing with
119262 ranges of symbolic names using position constant values which had been erroneously included
119263 in the final IEEE P1003.2b draft standard.

119264 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/14 is applied, correcting the example and
119265 adding a statement that the standard provides no means of defining a wide-character codeset.

119266 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/15 is applied, allowing the value zero for
119267 the width value of **WIDTH** and **WIDTH_DEFAULT**. This is required to cover some existing
119268 locales.

119269 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0031 [967] is applied.

119270 A.6.4.1 State-Dependent Character Encodings

119271 A requirement was considered that would force utilities to eliminate any redundant locking
119272 shifts, but this was left as a quality of implementation issue.

119273 This change satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex
119274 H.1:

119275 *The support of state-dependent (shift encoding) character sets should be addressed fully. See*
119276 *descriptions of these in XBD Section 6.2 (on page 128). If such character encodings are supported,*
119277 *it is expected that this will impact XBD Section 6.2 (on page 128), Chapter 7 (on page 135),*
119278 *Chapter 9 (on page 181), and the comm, cut, diff, grep, head, join, paste, and tail utilities.*

119279 The character set description file provides:

119280 The capability to describe character set attributes (such as collation order or character
119281 classes) independent of character set encoding, and using only the characters in the
119282 portable character set. This makes it possible to create generic *localedef* source files for all
119283 codesets that share the portable character set (such as the ISO 8859 family or IBM Extended
119284 ASCII).

119285 Standardized symbolic names for all characters in the portable character set, making it
119286 possible to refer to any such character regardless of encoding.

119287 Implementations are free to choose their own symbolic names, as long as the names identified
119288 by the Base Definitions volume of POSIX.1-2017 are also defined; this provides support for
119289 already existing “character names”.

119290 The names selected for the members of the portable character set follow the
119291 ISO/IEC 8859-1:1998 standard and the ISO/IEC 10646-1:2000 standard. However, several
119292 commonly used UNIX system names occur as synonyms in the list:

119293 The historical UNIX system names are used for control characters.

119294 The word “slash” is given in addition to “solidus”.

119295 The word “backslash” is given in addition to “reverse-solidus”.

119296 The word “hyphen” is given in addition to “hyphen-minus”.

119297 The word “period” is given in addition to “full-stop”.

119298 For digits, the word “digit” is eliminated.

119299 For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.

119300 The words “left brace” and “right brace” are given in addition to “left-curly-bracket” and
119301 “right-curly-bracket”.

119302 The names of the digits are preferred over the numbers to avoid possible confusion
119303 between '0' and 'o', and between '1' and 'l' (one and the letter ell).

119304 The names for the control characters in XBD Chapter 6 (on page 125) were taken from the
119305 ISO/IEC 4873:1991 standard.

119306 The charmap file was introduced to resolve problems with the portability of, especially, *localedef*
119307 sources. POSIX.1-2017 assumes that the portable character set is constant across all locales, but
119308 does not prohibit implementations from supporting two incompatible codings, such as both
119309 ASCII and EBCDIC. Such dual-support implementations should have all charmaps and *localedef*
119310 sources encoded using one portable character set, in effect cross-compiling for the other
119311 environment. Naturally, charmaps (and *localedef* sources) are only portable without
119312 transformation between systems using the same encodings for the portable character set. They

119313 can, however, be transformed between two sets using only a subset of the actual characters (the
 119314 portable character set). However, the particular coded character set used for an application or an
 119315 implementation does not necessarily imply different characteristics or collation; on the contrary,
 119316 these attributes should in many cases be identical, regardless of codeset. The charmap provides
 119317 the capability to define a common locale definition for multiple codesets (the same *localedef*
 119318 source can be used for codesets with different extended characters; the ability in the charmap to
 119319 define empty names allows for characters missing in certain codesets).

119320 The `<escape_char>` declaration was added at the request of the international community to ease
 119321 the creation of portable charmap files on terminals not implementing the default
 119322 `<backslash>`-escape. The `<comment_char>` declaration was added at the request of the
 119323 international community to eliminate the potential confusion between the `<number-sign>` and
 119324 the hash sign.

119325 The octal number notation with no leading zero required was selected to match those of *awk* and
 119326 *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant and
 119327 the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants must
 119328 contain at least two digits. As single-digit constants are relatively rare, this should not impose
 119329 any significant hardship. Provision is made for more digits to account for systems in which the
 119330 byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646-1:2000 standard) system
 119331 that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits.

119332 The decimal notation is supported because some newer international standards define character
 119333 values in decimal, rather than in the old column/row notation.

119334 The charmap identifies the coded character sets supported by an implementation. At least one
 119335 charmap must be provided, but no implementation is required to provide more than one.
 119336 Likewise, implementations can allow users to generate new charmaps (for instance, for a new
 119337 version of the ISO 8859 family of coded character sets), but does not have to do so. If users are
 119338 allowed to create new charmaps, the system documentation describes the rules that apply (for
 119339 instance, “only coded character sets that are supersets of the ISO/IEC 646:1991 standard IRV, no
 119340 multi-byte characters”).

119341 This addition of the **WIDTH** specification satisfies the following requirement from the
 119342 ISO POSIX-2:1993 standard, Annex H.1:

119343 (9) *The definition of column position relies on the implementation’s knowledge of the integral*
 119344 *width of the characters. The charmap or LC_CTYPE locale definitions should be enhanced to*
 119345 *allow application specification of these widths.*

119346 The character “width” information was first considered for inclusion under *LC_CTYPE* but was
 119347 moved because it is more closely associated with the information in the charmap than
 119348 information in the locale source (cultural conventions information). Concerns were raised that
 119349 formalizing this type of information is moving the locale source definition from the codeset-
 119350 independent entity that it was designed to be to a repository of codeset-specific information. A
 119351 similar issue occurred with the `<code_set_name>`, `<mb_cur_max>`, and `<mb_cur_min>`
 119352 information, which was resolved to reside in the charmap definition.

119353 The width definition was added to the IEEE P1003.2b draft standard with the intent that the
 119354 *wcswidth()* and/or *wcwidth()* functions (currently specified in the System Interfaces volume of
 119355 POSIX.1-2017) be the mechanism to retrieve the character width information.

119356 **A.7 Locale**

119357 **A.7.1 General**

119358 The description of locales is based on work performed in the UniForum Technical Committee,
119359 Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the
119360 ISO C standard or the X/Open Portability Guide.

119361 The value used to specify a locale with environment variables is the name specified as the *name*
119362 operand to the *localedef* utility when the locale was created. This provides a verifiable method to
119363 create and invoke a locale.

119364 The “object” definitions need not be portable, as long as “source” definitions are. Strictly
119365 speaking, source definitions are portable only between implementations using the same
119366 character set(s). Such source definitions, if they use symbolic names only, easily can be ported
119367 between systems using different codesets, as long as the characters in the portable character set
119368 (see XBD [Section 6.1](#), on page 125) have common values between the codesets; this is frequently
119369 the case in historical implementations. Of source, this requires that the symbolic names used for
119370 characters outside the portable character set be identical between character sets. The definition
119371 of symbolic names for characters is outside the scope of POSIX.1-2017, but is certainly within the
119372 scope of other standards organizations.

119373 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent)
119374 with the appropriate value. If the function is invoked with an empty string, the value of the
119375 corresponding environment variable is used. If the environment variable is not set or is set to the
119376 empty string, the implementation sets the appropriate environment as defined in XBD [Chapter 8](#)
119377 (on page 173).

119378 **A.7.2 POSIX Locale**

119379 On POSIX.1 implementations the POSIX locale is equal to the C locale, even though the
119380 requirements for the POSIX locale are more extensive than the ISO C standard requirements for
119381 the C locale. To avoid being classified as a C-language function, the name has been changed to
119382 the POSIX locale; the environment variable value can be either "POSIX" or, for historical
119383 reasons, "C".

119384 The POSIX definitions mirror the historical UNIX system behavior.

119385 The use of symbolic names for characters in the tables does not imply that the POSIX locale must
119386 be described using symbolic character names, but merely that it may be advantageous to do so.

119387 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0032 [796] and XBD/TC2-2008/0033
119388 [663] are applied.

119389 **A.7.3 Locale Definition**

119390 The decision to separate the file format from the *localedef* utility description was only partially
119391 editorial. Implementations may provide other interfaces than *localedef*. Requirements on “the
119392 utility”, mostly concerning error messages, are described in this way because they are meant to
119393 affect the other interfaces implementations may provide as well as *localedef*.

119394 The text about POSIX2_LOCALEDEF does not mean that internationalization is optional; only
119395 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize,
119396 for example, character class expressions such as “[[:alpha:]]”. A possible analogy is with
119397 an applications development environment; while all conforming implementations must be
119398 capable of executing applications, not all need to have the development environment installed.
119399 The assumption is that the capability to modify the behavior of utilities (and applications) via
119400 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to
119401 select an existing (presumably implementation-documented) locale. An implementation could,
119402 for example, choose to support only the POSIX locale, which would in effect limit the amount of
119403 changes from historical implementations quite drastically. The *localedef* utility is still required,
119404 but would always terminate with an exit code indicating that no locale could be created.
119405 Supported locales must be documented using the syntax defined in this chapter. (This ensures
119406 that users can accurately determine what capabilities are provided. If the implementation
119407 decides to provide additional capabilities to the ones in this chapter, that is already provided
119408 for.)

119409 If the option is present (that is, locales can be created), then the *localedef* utility must be capable
119410 of creating locales based on the syntax and rules defined in this chapter. This does not mean that
119411 the implementation cannot also provide alternate means for creating locales.

119412 The octal, decimal, and hexadecimal notations are the same employed by the charmap facility
119413 (see XBD Section 6.4, on page 129). To avoid confusion between an octal constant and a back-
119414 reference, the octal, hexadecimal, and decimal constants must contain at least two digits. As
119415 single-digit constants are relatively rare, this should not impose any significant hardship.
119416 Provision is made for more digits to account for systems in which the byte size is larger than 8
119417 bits. For example, a Unicode (see the ISO/IEC 10646-1:2000 standard) system that has defined
119418 16-bit bytes may require six octal, four hexadecimal, and five decimal digits. As with the
119419 charmap file, multi-byte characters are described in the locale definition file using “big-endian”
119420 notation for reasons of portability. There is no requirement that the internal representation in the
119421 computer memory be in this same order.

119422 One of the guidelines used for the development of this volume of POSIX.1-2017 is that
119423 characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in
119424 portable specifications. The <backslash> character is not in the invariant part; the <number-
119425 sign> is, but with multiple representations: as a <number-sign>, and as a hash sign. As far as
119426 general usage of these symbols, they are covered by the “grandfather clause”, but for newly
119427 defined interfaces, the WG15 POSIX working group has requested that POSIX provide alternate
119428 representations. Consequently, while the default escape character remains the <backslash> and
119429 the default comment character is the <number-sign>, implementations are required to recognize
119430 alternative representations, identified in the applicable source file via the <escape_char> and
119431 <comment_char> keywords.

119432 A.7.3.1 LC_CTYPE

119433 The `LC_CTYPE` category is primarily used to define the encoding-independent aspects of a
 119434 character set, such as character classification. In addition, certain encoding-dependent
 119435 characteristics are also defined for an application via the `LC_CTYPE` category. POSIX.1-2017 does
 119436 not mandate that the encoding used in the locale is the same as the one used by the application
 119437 because an implementation may decide that it is advantageous to define locales in a system-
 119438 wide encoding rather than having multiple, logically identical locales in different encodings, and
 119439 to convert from the application encoding to the system-wide encoding on usage. Other
 119440 implementations could require encoding-dependent locales.

119441 In either case, the `LC_CTYPE` attributes that are directly dependent on the encoding, such as
 119442 `<mb_cur_max>` and the display width of characters, are not user-specifiable in a locale source
 119443 and are consequently not defined as keywords.

119444 Implementations may define additional keywords or extend the `LC_CTYPE` mechanism to allow
 119445 application-defined keywords.

119446 The text “The ellipsis specification shall only be valid within a single encoded character set” is
 119447 present because it is possible to have a locale supported by multiple character encodings, as
 119448 explained in the rationale for XBD Section 6.1 (on page 125). An example given there is of a
 119449 possible Japanese-based locale supported by a mixture of the character sets JIS X 0201 Roman,
 119450 JIS X 0208, and JIS X 0201 Katakana. Attempting to express a range of characters across these sets
 119451 is not logical and the implementation is free to reject such attempts.

119452 As the `LC_CTYPE` character classes are based on the ISO C standard character class definition,
 119453 the category does not support multi-character elements. For instance, the German character
 119454 `<sharp-s>` is traditionally classified as a lowercase letter. There is no corresponding uppercase
 119455 letter; in proper capitalization of German text, the `<sharp-s>` will be replaced by “SS”; that is, by
 119456 two characters. This kind of conversion is outside the scope of the `toupper` and `tolower`
 119457 keywords.

119458 Where POSIX.1-2017 specifies that only certain characters can be specified, as for the keywords
 119459 `digit` and `xdigit`, the specified characters must be from the portable character set, as shown. As
 119460 an example, only the Arabic digits 0 through 9 are acceptable as digits.

119461 The character classes `digit`, `xdigit`, `lower`, `upper`, and `space` have a set of automatically included
 119462 characters. These only need to be specified if the character values (that is, encoding) differs from
 119463 the implementation default values. It is not possible to define a locale without these
 119464 automatically included characters unless some implementation extension is used to prevent
 119465 their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it
 119466 might not be possible for the standard utilities to be implemented as programs conforming to
 119467 the ISO C standard.

119468 The definition of character class `digit` requires that only ten characters †the ones defining
 119469 digits ‡can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here.
 119470 However, the encoding may vary if an implementation supports more than one encoding.

119471 The definition of character class `xdigit` requires that the characters included in character class
 119472 `digit` are included here also and allows for different symbols for the hexadecimal digits 10
 119473 through 15.

119474 The inclusion of the `charclass` keyword satisfies the following requirement from the
 119475 ISO POSIX-2: 1993 standard, Annex H.1:

119476 (3) *The `LC_CTYPE` (2.5.2.1) locale definition should be enhanced to allow user-specified additional*
 119477 *character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE)*
 119478 *`iswctype()` function.*

119479 This keyword was previously included in The Open Group specifications and is now mandated
119480 in the Shell and Utilities volume of POSIX.1-2017.

119481 The symbolic constant {CHARCLASS_NAME_MAX} was also adopted from The Open Group
119482 specifications. Applications portability is enhanced by the use of symbolic constants.

119483 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0033 [663], XBD/TC2-2008/0034 [663],
119484 XBD/TC2-2008/0035 [584], and XBD/TC2-2008/0036 [584] are applied.

119485 A.7.3.2 LC_COLLATE

119486 The rules governing collation depend to some extent on the use. At least five different levels of
119487 increasingly complex collation rules can be distinguished:

119488 1. *Byte/machine code order*: This is the historical collation order in the UNIX system and many
119489 proprietary operating systems. Collation is here performed character by character,
119490 without any regard to context. The primary virtue is that it usually is quite fast and also
119491 completely deterministic; it works well when the native machine collation sequence
119492 matches the user expectations.

119493 2. *Character order*: On this level, collation is also performed character by character, without
119494 regard to context. The order between characters is, however, not determined by the code
119495 values, but on the expectations by the user of the “correct” order between characters. In
119496 addition, such a (simple) collation order can specify that certain characters collate equally
119497 (for example, uppercase and lowercase letters).

119498 3. *String ordering*: On this level, entire strings are compared based on relatively
119499 straightforward rules. Several “passes” may be required to determine the order between
119500 two strings. Characters may be ignored in some passes, but not in others; the strings may
119501 be compared in different directions; and simple string substitutions may be performed
119502 before strings are compared. This level is best described as “dictionary” ordering; it is
119503 based on the spelling, not the pronunciation, or meaning, of the words.

119504 4. *Text search ordering*: This is a further refinement of the previous level, best described as
119505 “telephone book ordering”; some common homonyms (words spelled differently but
119506 with the same pronunciation) are collated together; numbers are collated as if they were
119507 spelled out, and so on.

119508 5. *Semantic-level ordering*: Words and strings are collated based on their meaning; entire
119509 words (such as “the”) are eliminated; the ordering is not deterministic. This usually
119510 requires special software and is highly dependent on the intended use.

119511 While the historical collation order formally is at level 1, for the English language it corresponds
119512 roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very
119513 much as it would be in a dictionary. While telephone book ordering would be an optimal goal
119514 for standard collation, this was ruled out as the order would be language-dependent.
119515 Furthermore, a requirement was that the order must be determined solely from the text string
119516 and the collation rules; no external information (for example, “pronunciation dictionaries”)
119517 could be required.

119518 As a result, the goal for the collation support is at level 3. This also matches the requirements for
119519 the Canadian collation order, as well as other, known collation requirements for alphabetic
119520 scripts. It specifically rules out collation based on pronunciation rules or based on semantic
119521 analysis of the text.

119522 The syntax for the *LC_COLLATE* category source meets the requirements for level 3 and has
119523 been verified to produce the correct result with examples based on French, Canadian, and
119524 Danish collation order. Because it supports multi-character collating elements, it is also capable

- 119525 of supporting collation in codesets where a character is expressed using non-spacing characters
119526 followed by the base character (such as the ISO/IEC 6937:2001 standard).
- 119527 The directives that can be specified in an operand to the **order_start** keyword are based on the
119528 requirements specified in several proposed standards and in customary use. The following is a
119529 rephrasing of rules defined for “lexical ordering in English and French” by the Canadian
119530 Standards Association (the text in square brackets is rephrased):
- 119531 Once special characters [punctuation] have been removed from original strings, the
119532 ordering is determined by scanning forwards (left to right) [disregarding case and
119533 diacriticals].
- 119534 In case of equivalence, special characters are once again removed from original strings and
119535 the ordering is determined by scanning backwards (starting from the rightmost character
119536 of the string and back), character by character [disregarding case but considering
119537 diacriticals].
- 119538 In case of repeated equivalence, special characters are removed again from original strings
119539 and the ordering is determined by scanning forwards, character by character [considering
119540 both case and diacriticals].
- 119541 If there is still an ordering equivalence after the first three rules have been applied, then
119542 only special characters and the position they occupy in the string are considered to
119543 determine ordering. The string that has a special character in the lowest position comes
119544 first. If two strings have a special character in the same position, the character [with the
119545 lowest collation value] comes first. In case of equality, the other special characters are
119546 considered until there is a difference or until all special characters have been exhausted.
- 119547 It is estimated that this part of POSIX.1-2017 covers the requirements for all European
119548 languages, and no particular problems are anticipated with Slavic or Middle East character sets.
- 119549 The Far East (particularly Japanese/Chinese) collations are often based on contextual
119550 information and pronunciation rules (the same ideogram can have different meanings and
119551 different pronunciations). Such collation, in general, falls outside the desired goal of
119552 POSIX.1-2017. There are, however, several other collation rules (stroke/radical or “most
119553 common pronunciation”) that can be supported with the mechanism described here.
- 119554 The character order is defined by the order in which characters and elements are specified
119555 between the **order_start** and **order_end** keywords. Weights assigned to the characters and
119556 elements define the collation sequence; in the absence of weights, the character order is also the
119557 collation sequence.
- 119558 The **position** keyword provides the capability to consider, in a compare, the relative position of
119559 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and
119560 "or-ing". Assuming the <hyphen-minus> is subject to **IGNORE** on the first pass, the two
119561 strings compare equal, and the position of the <hyphen-minus> is immaterial. On second pass,
119562 all characters except the <hyphen-minus> are subject to **IGNORE**, and in the normal case the
119563 two strings would again compare equal. By taking position into account, the first collates before
119564 the second.
- 119565 This standard recommends (by the use of “should” in the normative text) that all
119566 implementation-provided locales define a collation sequence that has a total ordering of all
119567 characters unless the locale name has an '@' modifier indicating that it has a special collation
119568 sequence. Defining locales in this way eliminates unexpected behavior when non-identical
119569 strings can collate equally (for example, `sort -u` and `sort | uniq` are not equivalent). The
119570 exception for locales with a suitable '@' modifier in the name allows implementations to supply
119571 locales which do not have a total ordering of all characters provided that they draw attention to
119572 it in the modifier name. For example, `@icase` could indicate that each upper and lowercase

119573 character pair collates equally. Even with an '@' modifier, total ordering is preferred when
 119574 possible; for example, characters that are "ignored" in dictionary order need not be completely
 119575 ignored (by using IGNORE for all collation weights), but can instead be given a unique weight
 119576 after one or more IGNORE weights.

119577 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0037 [938], XBD/TC2-2008/0038 [663],
 119578 and XBD/TC2-2008/0039 [584] are applied.

119579 A.7.3.3 LC_MONETARY

119580 The currency symbol does not appear in LC_MONETARY because it is not defined in the C
 119581 locale of the ISO C standard.

119582 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte
 119583 values. In locales based on multi-byte coded character sets, this cannot be enforced;
 119584 POSIX.1-2017 does not prohibit such characters, but makes the behavior unspecified (in the text
 119585 "In contexts where other standards ...").

119586 The grouping specification is based on, but not identical to, the ISO C standard. The -1 indicates
 119587 that no further grouping is performed; the equivalent of {CHAR_MAX} in the ISO C standard.

119588 The text "the value is not available in the locale" is taken from the ISO C standard and is used
 119589 instead of the "unspecified" text in early proposals. There is no implication that omitting these
 119590 keywords or assigning them values of "" or -1 produces unspecified results; such omissions or
 119591 assignments eliminate the effects described for the keyword or produce zero-length strings, as
 119592 appropriate.

119593 The locale definition is an extension of the ISO C standard localeconv() specification. In
 119594 particular, rules on how currency_symbol is treated are extended to also cover int_curr_symbol,
 119595 and p_sep_by_space and n_sep_by_space have been augmented with the value 2, which places
 119596 a <space> between the sign and the symbol. This has been updated to match the
 119597 ISO/IEC 9899:1999 standard requirements and is an incompatible change from UNIX 98 and the
 119598 ISO POSIX-2 standard and the ISO POSIX-1:1996 standard requirements. The following table
 119599 shows the result of various combinations:

		p_sep_by_space		
		2	1	0
p_cs_precedes = 1	p_sign_posn = 0	(\$1.25)	(\$ 1.25)	(\$1.25)
	p_sign_posn = 1	+ \$1.25	+\$ 1.25	+\$1.25
	p_sign_posn = 2	\$1.25 +	\$ 1.25+	\$1.25+
	p_sign_posn = 3	+ \$1.25	+\$ 1.25	+\$1.25
p_cs_precedes = 0	p_sign_posn = 4	\$ +1.25	+\$ 1.25	+\$1.25
	p_sign_posn = 0	(1.25 \$)	(1.25 \$)	(1.25\$)
	p_sign_posn = 1	+1.25 \$	+1.25 \$	+1.25\$
	p_sign_posn = 2	1.25\$ +	1.25 \$+	1.25\$+
	p_sign_posn = 3	1.25+ \$	1.25 +\$	1.25+\$
	p_sign_posn = 4	1.25\$ +	1.25 \$+	1.25\$+

119612 The following is an example of the interpretation of the mon_grouping keyword. Assuming that
 119613 the value to be formatted is 123 456 789 and the mon_thousands_sep is <apostrophe>, then the
 119614 following table shows the result. The third column shows the equivalent string in the ISO C
 119615 standard that would be used by the localeconv() function to accommodate this grouping.

	mon_grouping	Formatted Value	ISO C String
119616			
119617	3;-1	123456'789	"\3\177"
119618	3	123'456'789	"\3"
119619	3;2;-1	1234'56'789	"\3\2\177"
119620	3;2	12'34'56'789	"\3\2"
119621	-1	123456789	"\177"

119622 In these examples, the octal value of {CHAR_MAX} is 177.

119623 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/6 adds a correction that permits the Euro
119624 currency symbol and addresses extensibility. The correction is stated using the term "should"
119625 intentionally, in order to make this a recommendation rather than a restriction on
119626 implementations. This allows for flexibility in implementations on how they handle future
119627 currency symbol additions.

119628 IEEE Std 1003.1-2001/Cor 1-2002, tem XBD/TC1/D6/5 is applied, adding the `int_[np]_*` values
119629 to the POSIX locale definition of `LC_MONETARY`.

119630 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/16 is applied, updating the descriptions
119631 of `p_sep_by_space`, `n_sep_by_space`, `int_p_sep_by_space`, and `int_n_sep_by_space` to match
119632 the description of these keywords in the ISO C standard and the System Interfaces volume of
119633 POSIX.1-2017, `localeconv()`.

119634 A.7.3.4 `LC_NUMERIC`

119635 See the rationale for `LC_MONETARY` for a description of the behavior of grouping.

119636 A.7.3.5 `LC_TIME`

119637 Although certain of the conversion specifications in the POSIX locale (such as the name of the
119638 month) are shown with initial capital letters, this need not be the case in other locales. Programs
119639 using these conversion specifications may need to adjust the capitalization if the output is going
119640 to be used at the beginning of a sentence.

119641 The `LC_TIME` descriptions of `abday`, `day`, `mon`, and `abmon` imply a Gregorian style calendar
119642 (7-day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of
119643 calendars is outside the scope of POSIX.1-2017.

119644 While the ISO 8601:2004 standard numbers the weekdays starting with Monday, historical
119645 practice is to use the Sunday as the first day. Rather than change the order and introduce
119646 potential confusion, the days must be specified beginning with Sunday; previous references to
119647 "first day" have been removed. Note also that the Shell and Utilities volume of POSIX.1-2017
119648 `date` utility supports numbering compliant with the ISO 8601:2004 standard.

119649 As specified under `date` in the Shell and Utilities volume of POSIX.1-2017 and `strftime()` in the
119650 System Interfaces volume of POSIX.1-2017, the conversion specifications corresponding to the
119651 optional keywords consist of a modifier followed by a traditional conversion specification (for
119652 instance, `%Ex`). If the optional keywords are not supported by the implementation or are
119653 unspecified for the current locale, these modified conversion specifications are treated as the
119654 traditional conversion specifications. For example, assume the following keywords:

```
119655 alt_digits    "0th";"1st";"2nd";"3rd";"4th";"5th";\  
119656              "6th";"7th";"8th";"9th";"10th"
```

```
119657 d_fmt        "The %Od day of %B in %Y"
```

119658 On July 4th 1776, the `%x` conversion specifications would result in "The 4th day of July

119659 in 1776", while on July 14th 1789 it would result in "The 14 day of July in 1789". It
 119660 can be noted that the above example is for illustrative purposes only; the %O modifier is
 119661 primarily intended to provide for Kanji or Hindi digits in *date* formats.

119662 The following is an example for Japan that supports the current plus last three Emperors and
 119663 reverts to Western style numbering for years prior to the Meiji era. The example also allows for
 119664 the custom of using a special name for the first year of an era instead of using 1. (The examples
 119665 substitute romaji where kanji should be used.)

```
119666 era_d_fmt "%EY%mgatsu%dnichi (%a)"
119667 era      "+:2:1990/01/01:+*:Heisei:%EC%Eynen";\
119668         "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\
119669         "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eynen";\
119670         "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\
119671         "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eynen";\
119672         "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\
119673         "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eynen";\
119674         "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\
119675         "-:1868:1868/09/07:-*::%Ey"
```

119676 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield
 119677 the following results:

```
119678 %Ec - Heisei3nen9gatsu2lnichi (Sat) 14:39:26
119679 %EC - Heisei
119680 %Ex - Heisei3nen9gatsu2lnichi (Sat)
119681 %Ey - 3
119682 %EY - Heisei3nen
```

119683 Example era definitions for the Republic of China:

```
119684 era      "+:2:1913/01/01:+*:ChungHwaMingGuo:%EC%EyNen";\
119685         "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%ECYuenNen";\
119686         "+:1:1911/12/31:-*:MingChien:%EC%EyNen"
```

119687 Example definitions for the Christian Era:

```
119688 era      "+:1:0001/01/01:+*:AD:%EC %Ey";\
119689         "+:1:-0001/12/31:-*:BC:%Ey %EC"
```

119690 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0040 [912] is applied.

119691 A.7.3.6 LC_MESSAGES

119692 The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly
 119693 used to match user affirmative and negative responses. In POSIX.1-2017, the **yesexpr**, **noexpr**,
 119694 YESEXPR, and NOEXPR extended regular expressions have replaced them. Applications
 119695 should use the general locale-based messaging facilities to issue prompting messages which
 119696 include sample desired responses.

119697 Affirmative responses like:

```
119698 y
119699 Yes
119700 Yes!
```

119701 and negative responses like:

```
119702 N
```

119703 No
 119704 Never
 119705 No way!

119706 should all be recognized as affirmative and negative responses, respectively, by the EREs
 119707 identified by the **yesexpr** and **noexpr** keywords for English language-based locales. There is no
 119708 requirement that multi-line responses nor ambiguous responses like:

119709 no or yes
 119710 yes or no
 119711 maybe

119712 be correctly classified by either of these EREs. Application writers are encouraged to include
 119713 locale-specific suggestions for affirmative and negative responses in prompts.

119714 **A.7.4 Locale Definition Grammar**

119715 There is no additional rationale provided for this section.

119716 *A.7.4.1 Locale Lexical Conventions*

119717 There is no additional rationale provided for this section.

119718 *A.7.4.2 Locale Grammar*

119719 There is no additional rationale provided for this section.

119720 **A.7.5 Locale Definition Example**

119721 The following is an example of a locale definition file that could be used as input to the *localedef*
 119722 utility. It assumes that the utility is executed with the `-f` option, naming a charmap file with (at
 119723 least) the following content:

```

119724 CHARMAP
119725 <space>          \x20
119726 <dollar>         \x24
119727 <A>              \101
119728 <a>              \141
119729 <A-acute>        \346
119730 <a-acute>        \365
119731 <A-grave>        \300
119732 <a-grave>        \366
119733 <b>              \142
119734 <C>              \103
119735 <c>              \143
119736 <c-cedilla>      \347
119737 <d>              \x64
119738 <H>              \110
119739 <h>              \150
119740 <eszet>          \xb7
119741 <s>              \x73
119742 <z>              \x7a
119743 END CHARMAP

```

```

119744     It should not be taken as complete or to represent any actual locale, but only to illustrate the
119745     syntax.
119746     #
119747     LC_CTYPE
119748     lower  <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
119749     upper  A;B;C;Ç;...;Z
119750     space  \x20;\x09;\x0a;\x0b;\x0c;\x0d
119751     blank  \040;\011
119752     toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
119753     END LC_CTYPE
119754     #
119755     LC_COLLATE
119756     #
119757     # The following example of collation is based on
119758     # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
119759     # Ordering Standard for Character Sets of CSA Z234.4 Standard".
119760     # (Other parts of this example locale definition file do not
119761     # purport to relate to Canada, or to any other real culture.)
119762     # The proposed standard defines a 4-weight collation, such that
119763     # in the first pass, characters are compared without regard to
119764     # case or accents; in the second pass, backwards-compare without
119765     # regard to case; in the third pass, forwards-compare without
119766     # regard to diacriticals. In the 3 first passes, non-alphabetic
119767     # characters are ignored; in the fourth pass, only special
119768     # characters are considered, such that "The string that has a
119769     # special character in the lowest position comes first. If two
119770     # strings have a special character in the same position, the
119771     # collation value of the special character determines ordering.
119772     #
119773     # Only a subset of the character set is used here; mostly to
119774     # illustrate the set-up.
119775     #
119776     collating-symbol <NULL>
119777     collating-symbol <LOW_VALUE>
119778     collating-symbol <LOWER-CASE>
119779     collating-symbol <SUBSCRIPT-LOWER>
119780     collating-symbol <SUPERSCRIPT-LOWER>
119781     collating-symbol <UPPER-CASE>
119782     collating-symbol <NO-ACCENT>
119783     collating-symbol <PECULIAR>
119784     collating-symbol <LIGATURE>
119785     collating-symbol <ACUTE>
119786     collating-symbol <GRAVE>
119787     # Further collating-symbols follow.
119788     #
119789     # Properly, the standard does not include any multi-character
119790     # collating elements; the one below is added for completeness.
119791     #
119792     collating_element <ch> from "<c><h>"
119793     collating_element <CH> from "<C><H>"
119794     collating_element <Ch> from "<C><h>"
119795     #

```

```

119796     order_start forward;backward;forward;forward,position
119797     #
119798     # Collating symbols are specified first in the sequence to allocate
119799     # basic collation values to them, lower than that of any character.
119800     <NULL>
119801     <LOW_VALUE>
119802     <LOWER-CASE>
119803     <SUBSCRIPT-LOWER>
119804     <SUPERSCRIPT-LOWER>
119805     <UPPER-CASE>
119806     <NO-ACCENT>
119807     <PECULIAR>
119808     <LIGATURE>
119809     <ACUTE>
119810     <GRAVE>
119811     <RING-ABOVE>
119812     <DIAERESIS>
119813     <TILDE>
119814     # Further collating symbols are given a basic collating value here.
119815     #
119816     # Here follow special characters.
119817     <space>          IGNORE;IGNORE;IGNORE;<space>
119818     # Other special characters follow here.
119819     #
119820     # Here follow the regular characters.
119821     <a>              <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
119822     <A>              <a>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
119823     <a-acute>        <a>;<ACUTE>;<LOWER-CASE>;IGNORE
119824     <A-acute>        <a>;<ACUTE>;<UPPER-CASE>;IGNORE
119825     <a-grave>        <a>;<GRAVE>;<LOWER-CASE>;IGNORE
119826     <A-grave>        <a>;<GRAVE>;<UPPER-CASE>;IGNORE
119827     <ae>            "<a><e>";"<LIGATURE><LIGATURE>";\
119828                  "<LOWER-CASE><LOWER-CASE>";IGNORE
119829     <AE>            "<a><e>";"<LIGATURE><LIGATURE>";\
119830                  "<UPPER-CASE><UPPER-CASE>";IGNORE
119831     <b>              <b>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
119832     <B>              <b>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
119833     <c>              <c>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
119834     <C>              <c>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
119835     <ch>            <ch>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
119836     <Ch>            <ch>;<NO-ACCENT>;<PECULIAR>;IGNORE
119837     <CH>            <ch>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
119838     #
119839     # As an example, the strings "Bach" and "bach" could be encoded (for
119840     # compare purposes) as:
119841     # "Bach" <b>;<a>;<ch>;<LOW_VALUE>;<NO-ACCENT>;<NO-ACCENT>;\
119842     #       <NO-ACCENT>;<LOW_VALUE>;<UPPER-CASE>;<LOWER-CASE>;\
119843     #       <LOWER-CASE>;<NULL>
119844     # "bach" <b>;<a>;<ch>;<LOW_VALUE>;<NO-ACCENT>;<NO-ACCENT>;\
119845     #       <NO-ACCENT>;<LOW_VALUE>;<LOWER-CASE>;<LOWER-CASE>;\
119846     #       <LOWER-CASE>;<NULL>
119847     #
119848     # The two strings are equal in pass 1 and 2, but differ in pass 3.

```

```

119849      #
119850      # Further characters follow.
119851      #
119852      UNDEFINED      IGNORE;IGNORE;IGNORE;IGNORE
119853      #
119854      order_end
119855      #
119856      END LC_COLLATE
119857      #
119858      LC_MONETARY
119859      int_curr_symbol      "USD "
119860      currency_symbol      "$"
119861      mon_decimal_point    "."
119862      mon_grouping         3;0
119863      positive_sign        ""
119864      negative_sign        "-"
119865      p_cs_precedes        1
119866      n_sign_posn          0
119867      END LC_MONETARY
119868      #
119869      LC_NUMERIC
119870      copy "US_en.ASCII"
119871      END LC_NUMERIC
119872      #
119873      LC_TIME
119874      abday "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
119875      #
119876      day "Sunday";"Monday";"Tuesday";"Wednesday";\
119877      "Thursday";"Friday";"Saturday"
119878      #
119879      abmon "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
119880      "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
119881      #
119882      mon "January";"February";"March";"April";\
119883      "May";"June";"July";"August";"September";\
119884      "October";"November";"December"
119885      #
119886      d_t_fmt "%a %b %d %T %Z %Y\n"
119887      END LC_TIME
119888      #
119889      LC_MESSAGES
119890      yesexpr "^[yY][[:alpha:]]*"|(OK)"
119891      #
119892      noexpr "[nN][[:alpha:]]*"
119893      END LC_MESSAGES

```

119894 **A.8 Environment Variables**

119895 **A.8.1 Environment Variable Definition**

119896 The variable *environ* is not intended to be declared in any header, but rather to be declared by
 119897 the user for accessing the array of strings that is the environment. This is the traditional usage of
 119898 the symbol. Putting it into a header could break some programs that use the symbol for their
 119899 own purposes.

119900 The decision to restrict conforming systems to the use of digits, uppercase letters, and
 119901 underscores for environment variable names allows applications to use lowercase letters in their
 119902 environment variable names without conflicting with any conforming system.

119903 In addition to the obvious conflict with the shell syntax for positional parameter substitution,
 119904 some historical applications (including some shells) exclude names with leading digits from the
 119905 environment.

119906 **A.8.2 Internationalization Variables**

119907 Utilities conforming to the Shell and Utilities volume of POSIX.1-2017 and written in standard C
 119908 can access the locale variables by issuing the following call:

```
119909 setlocale(LC_ALL, "")
```

119910 If this were omitted, the ISO C standard specifies that the C (or POSIX) locale would be used.

119911 The DESCRIPTION of *setlocale()* requires that when setting all categories of a locale, if the value
 119912 of any of the environment variable searches yields a locale that is not supported (and non-null),
 119913 the *setlocale()* function returns a null pointer and the global locale is unchanged.

119914 For the standard utilities, if any of the environment variables are invalid, it makes sense to
 119915 default to an implementation-defined, consistent locale environment. It is more confusing for a
 119916 user to have partial settings occur in case of a mistake. All utilities would then behave in one
 119917 language/cultural environment. Furthermore, it provides a way of forcing the whole
 119918 environment to be the implementation-defined default. Disastrous results could occur if a
 119919 pipeline of utilities partially uses the environment variables in different ways. In this case, it
 119920 would be appropriate for utilities that use *LANG* and related variables to exit with an error if
 119921 any of the variables are invalid. For example, users typing individual commands at a terminal
 119922 might want *date* to work if *LC_MONETARY* is invalid as long as *LC_TIME* is valid. Since these
 119923 are conflicting reasonable alternatives, POSIX.1-2017 leaves the results unspecified if the locale
 119924 environment variables would not produce a complete locale matching the specification of the
 119925 user.

119926 The locale settings of individual categories cannot be truly independent and still guarantee
 119927 correct results. For example, when collating two strings, characters must first be extracted from
 119928 each string (governed by *LC_CTYPE*) before being mapped to collating elements (governed by
 119929 *LC_COLLATE*) for comparison. That is, if *LC_CTYPE* is causing parsing according to the rules of
 119930 a large, multi-byte code set (potentially returning 20000 or more distinct character codeset
 119931 values), but *LC_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters,
 119932 meaningful results are obviously impossible.

119933 The *LC_MESSAGES* variable affects the language of messages generated by the standard
 119934 utilities.

119935 The description of the environment variable names starting with the characters ``LC_”
 119936 acknowledges the fact that the interfaces presented may be extended as new international
 119937 functionality is required. In the ISO C standard, names preceded by ``LC_” are reserved in the
 119938 name space for future categories.

119939 To avoid name clashes, new categories and environment variables are divided into two
 119940 classifications: ``implementation-independent” and ``implementation-defined”.

119941 Implementation-independent names will have the following format:

119942 `LC_NAME`

119943 where *NAME* is the name of the new category and environment variable. Capital letters must be
 119944 used for implementation-independent names.

119945 Implementation-defined names must be in lowercase letters, as below:

119946 `LC_name`

119947 **A.8.3 Other Environment Variables**

119948 **COLUMNS, LINES**

119949 The default values for the number of column positions, *COLUMNS*, and screen height, *LINES*,
 119950 are unspecified because historical implementations use different methods to determine values
 119951 corresponding to the size of the screen in which the utility is run. This size is typically known to
 119952 the implementation through the value of *TERM*, or by more elaborate methods such as
 119953 extensions to the *stty* utility or knowledge of how the user is dynamically resizing windows on a
 119954 bit-mapped display terminal. Users should not need to set these variables in the environment
 119955 unless there is a specific reason to override the default behavior of the implementation, such as
 119956 to display data in an area arbitrarily smaller than the terminal or window. Values for these
 119957 variables that are not decimal integers greater than zero are implicitly undefined values; it is
 119958 unnecessary to enumerate all of the possible values outside of the acceptable set.

119959 **LOGNAME**

119960 In most implementations, the value of such a variable is easily forged, so security-critical
 119961 applications should rely on other means of determining user identity. *LOGNAME* is required to
 119962 be constructed from the portable filename character set for reasons of interchange. No diagnostic
 119963 condition is specified for violating this rule, and no requirement for enforcement exists. The
 119964 intent of the requirement is that if extended characters are used, the ``guarantee” of portability
 119965 implied by a standard is void.

119966 **PATH**

119967 Many historical implementations of the Bourne shell do not interpret a trailing <colon> to
 119968 represent the current working directory and are thus non-conforming. The C Shell and the
 119969 KornShell conform to POSIX.1-2017 on this point. The usual name of dot may also be used to
 119970 refer to the current working directory.

119971 Many implementations historically have used a default value of */bin* and */usr/bin* for the *PATH*
 119972 variable. POSIX.1-2017 does not mandate this default path be identical to that retrieved from
 119973 *getconf PATH* because it is likely that the standardized utilities may be provided in another
 119974 directory separate from the directories used by some historical applications.

119975 **SHELL**

119976 The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is
 119977 no direct requirement that that shell conform to POSIX.1-2017; that decision should rest with the
 119978 user. It is the intention of the standard developers that alternative shells be permitted, if the user
 119979 chooses to develop or acquire one. An operating system that builds its shell into the “kernel” in
 119980 such a manner that alternative shells would be impossible does not conform to the spirit of
 119981 POSIX.1-2017.

119982 **TZ**

119983 The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any
 119984 name that contains the <plus-sign> ('+'), the <hyphen-minus> ('-'), or digits), which may be
 119985 appropriate for countries that do not have an official timezone name. It would be coded as
 119986 <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of
 119987 UTC+2, each with a length of 5 characters. This does not appear to conflict with any existing
 119988 usage. The characters '<' and '>' were chosen for quoting because they are easier to parse
 119989 visually than a quoting character that does not provide some sense of bracketing (and in a string
 119990 like this, such bracketing is helpful). They were also chosen because they do not need special
 119991 treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a
 119992 string. Because '<' and '>' are meaningful to the shell, the whole string would have to be
 119993 quoted, but that is easily explained. (Parentheses would have presented the same problems.)
 119994 Although the '>' symbol could have been permitted in the string by either escaping it or
 119995 doubling it, it seemed of little value to require that. This could be provided as an extension if
 119996 there was a need. Timezone names of this new form lead to a requirement that the value of
 119997 {_POSIX_TZNAME_MAX} change from 3 to 6.

119998 Since the *TZ* environment variable is usually inherited by all applications started by a user after
 119999 the value of the *TZ* environment variable is changed and since many applications run using the
 120000 C or POSIX locale, using characters that are not in the portable character set in the *std* and *dst*
 120001 fields could cause unexpected results.

120002 The format of the *TZ* environment variable is changed in Issue 6 to allow for the quoted form, as
 120003 defined in earlier versions of the ISO POSIX-1 standard.

120004 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/7 is applied, adding the *ctime_r()* and
 120005 *localtime_r()* functions to the list of functions that use the *TZ* environment variable.

120006 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0041 [584] is applied.

120007 **A.9 Regular Expressions**

120008 Rather than repeating the description of REs for each utility supporting REs, the standard
 120009 developers preferred a common, comprehensive description of regular expressions in one place.
 120010 The most common behavior is described here, and exceptions or extensions to this are
 120011 documented for the respective utilities, as appropriate.

120012 The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical
 120013 *egrep* type (now *grep -E*).

120014 The text is based on the *ed* description and substantially modified, primarily to aid developers
 120015 and others in the understanding of the capabilities and limitations of REs. Much of this was
 120016 influenced by internationalization requirements.

120017 It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does
 120018 not employ REs.

120019 The specification of REs is particularly important to internationalization because pattern
 120020 matching operations are very basic operations in business and other operations. The syntax and
 120021 rules of REs are intended to be as intuitive as possible to make them easy to understand and use.
 120022 The historical rules and behavior do not provide that capability to non-English language users,
 120023 and do not provide the necessary support for commonly used characters and language
 120024 constructs. It was necessary to provide extensions to the historical RE syntax and rules to
 120025 accommodate other languages.

120026 As they are limited to bracket expressions, the rationale for these modifications is in XBD [Section](#)
 120027 [9.3.5](#) (on page 184).

120028 **A.9.1 Regular Expression Definitions**

120029 It is possible to determine what strings correspond to subexpressions by recursively applying
 120030 the leftmost longest rule to each subexpression, but only with the proviso that the overall match
 120031 is leftmost longest. For example, matching "`\(ac*\\)c*d[ac]*\1`" against `acdacaaa` matches
 120032 `acdacaaa` (with `\1=a`); simply matching the longest match for "`\(ac*\\)`" would yield `\1=ac`, but
 120033 the overall match would be smaller (`acdac`). Conceptually, the implementation must examine
 120034 every possible match and among those that yield the leftmost longest total matches, pick the one
 120035 that does the longest match for the leftmost subexpression, and so on. Note that this means that
 120036 matching by subexpressions is context-dependent: a subexpression within a larger RE may
 120037 match a different string from the one it would match as an independent RE, and two instances of
 120038 the same subexpression within the same larger RE may match different lengths even in similar
 120039 sequences of characters. For example, in the ERE "`(a.*b)(a.*b)`", the two identical
 120040 subexpressions would match four and six characters, respectively, of `acbbaccccb`.

120041 The definition of single character has been expanded to include also collating elements
 120042 consisting of two or more characters; this expansion is applicable only when a bracket
 120043 expression is included in the BRE or ERE. An example of such a collating element may be the
 120044 Dutch *ij*, which collates as a 'y'. In some encodings, a ligature "i with j" exists as a character
 120045 and would represent a single-character collating element. In another encoding, no such ligature
 120046 exists, and the two-character sequence *ij* is defined as a multi-character collating element.
 120047 Outside brackets, the *ij* is treated as a two-character RE and matches the same characters in a
 120048 string. Historically, a bracket expression only matched a single character. The ISO POSIX-2: 1993
 120049 standard required bracket expressions like "`^[[:lower:]]`" to match multi-character collating
 120050 elements such as "`ij`". However, this requirement led to behavior that many users did not
 120051 expect and that could not feasibly be mimicked in user code, and it was rarely if ever
 120052 implemented correctly. The current standard leaves it unspecified whether a bracket expression
 120053 matches a multi-character collating element, allowing both historical and ISO POSIX-2: 1993
 120054 standard implementations to conform.

120055 Also, in the current standard, it is unspecified whether character class expressions like
 120056 "`[[:lower:]]`" can include multi-character collating elements like "`ij`"; hence
 120057 "`[[:lower:]]`" can match "`ij`", and "`^[[:lower:]]`" can fail to match "`ij`". Common
 120058 practice is for a character class expression to match a collating element if it matches the collating
 120059 element's first character.

120060 **A.9.2 Regular Expression General Requirements**

120061 The definition of which sequence is matched when several are possible is based on the leftmost-
 120062 longest rule historically used by deterministic recognizers. This rule is easier to define and
 120063 describe, and arguably more useful, than the first-match rule historically used by non-
 120064 deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully
 120065 contrived examples are needed to demonstrate the difference.

120066 A formal expression of the leftmost-longest rule is:

120067 The search is performed as if all possible suffixes of the string were tested for a prefix
 120068 matching the pattern; the longest suffix containing a matching prefix is chosen, and the
 120069 longest possible matching prefix of the chosen suffix is identified as the matching
 120070 sequence.

120071 Historically, most RE implementations only match lines, not strings. However, that is more an
 120072 effect of the usage than of an inherent feature of REs themselves. Consequently, POSIX.1-2017
 120073 does not regard <newline> characters as special; they are ordinary characters, and both a
 120074 <period> and a non-matching list can match them. Those utilities (like *grep*) that do not allow
 120075 <newline> characters to match are responsible for eliminating any <newline> from strings
 120076 before matching against the RE. The *regcomp()* function, however, can provide support for such
 120077 processing without violating the rules of this section.

120078 Some implementations of *egrep* have had very limited flexibility in handling complex EREs.
 120079 POSIX.1-2017 does not attempt to define the complexity of a BRE or ERE, but does place a lower
 120080 limit on it—any RE must be handled, as long as it can be expressed in 256 bytes or less. (Of
 120081 course, this does not place an upper limit on the implementation.) There are historical programs
 120082 using a non-deterministic-recognizer implementation that should have no difficulty with this
 120083 limit. It is possible that a good approach would be to attempt to use the faster, but more limited,
 120084 deterministic recognizer for simple expressions and to fall back on the non-deterministic
 120085 recognizer for those expressions requiring it. Non-deterministic implementations must be
 120086 careful to observe the rules on which match is chosen; the longest match, not the first match,
 120087 starting at a given character is used.

120088 The term “invalid” highlights a difference between this section and some others: POSIX.1-2017
 120089 frequently avoids mandating of errors for syntax violations because they can be used by
 120090 implementors to trigger extensions. However, the authors of the internationalization features of
 120091 REs wanted to mandate errors for certain conditions to identify usage problems or non-portable
 120092 constructs. These are identified within this rationale as appropriate. The remaining syntax
 120093 violations have been left implicitly or explicitly undefined. For example, the BRE construct
 120094 “\{1,2,3\}” does not comply with the grammar. A conforming application cannot rely on it
 120095 producing an error nor matching the literal characters “\{1,2,3\}”.

120096 The term “undefined” was used in favor of “unspecified” because many of the situations are
 120097 considered errors on some implementations, and the standard developers considered that
 120098 consistency throughout the section was preferable to mixing undefined and unspecified.

120099 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0042 [554] is applied.

120100 **A.9.3 Basic Regular Expressions**

120101 There is no additional rationale provided for this section.

120102 *A.9.3.1 BREs Matching a Single Character or Collating Element*

120103 There is no additional rationale provided for this section.

120104 *A.9.3.2 BRE Ordinary Characters*

120105 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0043 [554] is applied.

120106 *A.9.3.3 BRE Special Characters*

120107 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0043 [554] is applied.

120108 *A.9.3.4 Periods in BREs*

120109 There is no additional rationale provided for this section.

120110 *A.9.3.5 RE Bracket Expression*

120111 Range expressions are, historically, an integral part of REs. However, the requirements of
 120112 “natural language behavior” and portability do conflict. In the POSIX locale, ranges must be
 120113 treated according to the collating sequence and include such characters that fall within the range
 120114 based on that collating sequence, regardless of character values. In other locales, ranges have
 120115 unspecified behavior.

120116 Some historical implementations allow range expressions where the ending range point of one
 120117 range is also the starting point of the next (for instance, "[a-m-o]"). This behavior should not
 120118 be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is
 120119 a valid expression and how it should be interpreted.

120120 Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per XBD
 120121 [Table 5-1](#) (on page 121), while the normal ERE behavior is to regard such a sequence as
 120122 consisting of two characters. Allowing the *awk/lex* behavior in EREs would change the normal
 120123 behavior in an unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in
 120124 EREs before passing them to *regcomp()* or comparable routines. Each utility describes the escape
 120125 sequences it accepts as an exception to the rules in this section; the list is not the same, for
 120126 historical reasons.

120127 As noted previously, the new syntax and rules have been added to accommodate other
 120128 languages than English. The remainder of this section describes the rationale for these
 120129 modifications.

120130 In the POSIX locale, a regular expression that starts with a range expression matches a set of
 120131 strings that are contiguously sorted, but this is not necessarily true in other locales. For example,
 120132 a French locale might have the following behavior:

```
120133 $ ls
120134 alpha  Alpha  estim  ESTIM  t  eurka
120135 $ ls [a-e]*
120136 alpha  Alpha  estim  eurka
```

120137 Such disagreements between matching and contiguous sorting are unavoidable because POSIX

120138 sorting cannot be implemented in terms of a deterministic finite-state automaton (DFA), but
120139 range expressions by design are implementable in terms of DFAs.

120140 Historical implementations used native character order to interpret range expressions. The
120141 ISO POSIX-2:1993 standard instead required collating element order (CEO): the order that
120142 collating elements were specified between the **order_start** and **order_end** keywords in the
120143 *LC_COLLATE* category of the current locale. CEO had some advantages in portability over the
120144 native character order, but it also had some disadvantages:

120145 CEO could not feasibly be mimicked in user code, leading to inconsistencies between
120146 POSIX matchers and matchers in popular user programs like Emacs, *ksh*, and Perl.

120147 CEO caused range expressions to match accented and capitalized letters contrary to many
120148 users' expectations. For example, "[a-e]" typically matched both 'E' and ' ' but
120149 neither 'A' nor ' ' .

120150 CEO was not consistent across implementations. In practice, CEO was often less portable
120151 than native character order. For example, it was common for the CEOs of two
120152 implementation-supplied locales to disagree, even if both locales were named "da_DK".

120153 Because of these problems, some implementations of regular expressions continued to use native
120154 character order. Others used the collation sequence, which is more consistent with sorting than
120155 either CEO or native order, but which departs further from the traditional POSIX semantics
120156 because it generally requires "[a-e]" to match either 'A' or 'E' but not both. As a result of
120157 this kind of implementation variation, programmers who wanted to write portable regular
120158 expressions could not rely on the ISO POSIX-2:1993 standard guarantees in practice.

120159 While revising the standard, lengthy consideration was given to proposals to attack this problem
120160 by adding an API for querying the CEO to allow user-mode matchers, but none of these
120161 proposals had implementation experience and none achieved consensus. Leaving the standard
120162 alone was also considered, but rejected due to the problems described above.

120163 The current standard leaves unspecified the behavior of a range expression outside the POSIX
120164 locale. This makes it clearer that conforming applications should avoid range expressions
120165 outside the POSIX locale, and it allows implementations and compatible user-mode matchers to
120166 interpret range expressions using native order, CEO, collation sequence, or other, more
120167 advanced techniques. The concerns which led to this change were raised in IEEE PASC
120168 interpretation 1003.2 #43 and others, and related to ambiguities in the specification of how
120169 multi-character collating elements should be handled in range expressions. These ambiguities
120170 had led to multiple interpretations of the specification, in conflicting ways, which led to varying
120171 implementations. As noted above, efforts were made to resolve the differences, but no solution
120172 has been found that would be specific enough to allow for portable software while not
120173 invalidating existing implementations.

120174 The standard developers recognize that collating elements are important, such elements being
120175 common in several European languages; for example, 'ch' or 'll' in traditional Spanish;
120176 'aa' in several Scandinavian languages. Existing internationalized implementations have
120177 processed, and continue to process, these elements in range expressions. Efforts are expected to
120178 continue in the future to find a way to define the behavior of these elements precisely and
120179 portably.

120180 The ISO POSIX-2:1993 standard required "[b-a]" to be an invalid expression in the POSIX
120181 locale, but this requirement has been relaxed in this version of the standard so that "[b-a]" can
120182 instead be treated as a valid expression that does not match any string.

120183 The standard specifies three possible behaviors for regular expressions such as "[:alpha:]".
120184 One behavior is the traditional implementation, which behaves like "[:ahlp]". Another, for
120185 alignment with the *tr* utility, is to treat it like "[[:alpha:]]". And finally, the standard allows

120186 rejecting the regular expression as invalid, as a means of alerting a user to the non-portable
 120187 aspect of that regular expression. The set of regular expressions with this undefined behavior is
 120188 limited solely to the expressions where the outer '[' and ']' of the bracket expression can be
 120189 confused with the missing bracket pair '[' and ']' necessary to form a collating symbol,
 120190 equivalence class, or character class; thus "[_alpha:]" or "[::]" do not trigger the
 120191 unspecified behavior.

120192 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0044 [938], XBD/TC2-2008/0045 [872],
 120193 XBD/TC2-2008/0046 [938], XBD/TC2-2008/0047 [584], and XBD/TC2-2008/0048 [584] are
 120194 applied.

120195 A.9.3.6 BREs Matching Multiple Characters

120196 The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit
 120197 identifier; increasing this to multiple digits would break historical applications. This does not
 120198 imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten
 120199 subexpressions:

```
120200 \(\(\(ab\) *c\) *d\)\(ef\) *\(\(gh\)\{2\}\(ij\) *\(\(kl\) *\(\(mn\) *\(\(op\) *\(\(qr\) *
```

120201 The standard developers regarded the common historical behavior, which supported "\n*", but
 120202 not "\n\{min,max\}", "\(\dots\) *", or "\(\dots\)\{min,max\}", as a non-intentional
 120203 result of a specific implementation, and they supported both duplication and interval
 120204 expressions following subexpressions and back-references.

120205 The changes to the processing of the back-reference expression remove an unspecified or
 120206 ambiguous behavior in the Shell and Utilities volume of POSIX.1-2017, aligning it with the
 120207 requirements specified for the *regcomp()* expression, and is the result of PASC Interpretation
 120208 1003.2-92 #43 submitted for the ISO POSIX-2: 1993 standard.

120209 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0049 [595] is applied.

120210 A.9.3.7 BRE Precedence

120211 There is no additional rationale provided for this section.

120212 A.9.3.8 BRE Expression Anchoring

120213 Often, the <dollar-sign> is viewed as matching the ending <newline> in text files. This is not
 120214 strictly true; the <newline> is typically eliminated from the strings to be matched, and the
 120215 <dollar-sign> matches the terminating null character.

120216 The ability of '^', '\$', and '*' to be non-special in certain circumstances may be confusing to
 120217 some programmers, but this situation was changed only in a minor way from historical practice
 120218 to avoid breaking many historical scripts. Some consideration was given to making the use of
 120219 the anchoring characters undefined if not escaped and not at the beginning or end of strings.
 120220 This would cause a number of historical BREs, such as "2^10", "\$HOME", and "\$1.35", that
 120221 relied on the characters being treated literally, to become invalid.

120222 However, one relatively uncommon case was changed to allow an extension used on some
 120223 implementations. Historically, the BREs "\foo" and "\(^foo\)" did not match the same
 120224 string, despite the general rule that subexpressions and entire BREs match the same strings. To
 120225 increase consensus, POSIX.1-2017 has allowed an extension on some implementations to treat
 120226 these two cases in the same way by declaring that anchoring *may* occur at the beginning or end
 120227 of a subexpression. Therefore, portable BREs that require a literal <circumflex> at the beginning
 120228 or a <dollar-sign> at the end of a subexpression must escape them. Note that a BRE such as

120229 "a\ (^bc\)" will either match "a^bc" or nothing on different systems under the rules.

120230 ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped
 120231 anchor character has never matched its literal counterpart outside a bracket expression. Some
 120232 implementations treated "foo\$bar" as a valid expression that never matched anything; others
 120233 treated it as invalid. POSIX.1-2017 mandates the former, valid unmatched behavior.

120234 Some implementations have extended the BRE syntax to add alternation. For example, the
 120235 subexpression "\ (foo\$\ |bar\)" would match either "foo" at the end of the string or "bar"
 120236 anywhere. The extension is triggered by the use of the undefined "\|" sequence. Because the
 120237 BRE is undefined for portable scripts, the extending system is free to make other assumptions,
 120238 such that the '\$' represents the end-of-line anchor in the middle of a subexpression. If it were
 120239 not for the extension, the '\$' would match a literal <dollar-sign> under the rules.

120240 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0049 [595] is applied.

120241 A.9.4 Extended Regular Expressions

120242 As with BREs, the standard developers decided to make the interpretation of escaped ordinary
 120243 characters undefined.

120244 The <right-parenthesis> is not listed as an ERE special character because it is only special in the
 120245 context of a preceding <left-parenthesis>. If found without a preceding <left-parenthesis>, the
 120246 <right-parenthesis> has no special meaning.

120247 The interval expression, "{m,n}", has been added to EREs. Historically, the interval expression
 120248 has only been supported in some ERE implementations. The standard developers estimated that
 120249 the addition of interval expressions to EREs would not decrease consensus and would also make
 120250 BREs more of a subset of EREs than in many historical implementations.

120251 It was suggested that, in addition to interval expressions, back-references ('\n') should also be
 120252 added to EREs. This was rejected by the standard developers as likely to decrease consensus.

120253 In historical implementations, multiple duplication symbols are usually interpreted from left to
 120254 right and treated as additive. As an example, "a+b" matches zero or more instances of 'a'
 120255 followed by a 'b'. In POSIX.1-2017, multiple duplication symbols are undefined; that is, they
 120256 cannot be relied upon for conforming applications. One reason for this is to provide some scope
 120257 for future enhancements.

120258 The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons,
 120259 interval expressions have a lower precedence than concatenation.

120260 A.9.4.1 EREs Matching a Single Character or Collating Element

120261 There is no additional rationale provided for this section.

120262 A.9.4.2 ERE Ordinary Characters

120263 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0050 [554] is applied.

- 120264 A.9.4.3 *ERE Special Characters*
 120265 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0050 [554] is applied.
- 120266 A.9.4.4 *Periods in EREs*
 120267 There is no additional rationale provided for this section.
- 120268 A.9.4.5 *ERE Bracket Expression*
 120269 There is no additional rationale provided for this section.
- 120270 A.9.4.6 *EREs Matching Multiple Characters*
 120271 There is no additional rationale provided for this section.
- 120272 A.9.4.7 *ERE Alternation*
 120273 There is no additional rationale provided for this section.
- 120274 A.9.4.8 *ERE Precedence*
 120275 There is no additional rationale provided for this section.
- 120276 A.9.4.9 *ERE Expression Anchoring*
 120277 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0051 [595] is applied.

120278 **A.9.5 Regular Expression Grammar**

- 120279 The grammars are intended to represent the range of acceptable syntaxes available to
 120280 conforming applications. There are instances in the text where undefined constructs are
 120281 described; as explained previously, these allow implementation extensions. There is no intended
 120282 requirement that an implementation extension must somehow fit into the grammars shown
 120283 here.
- 120284 The BRE grammar does not permit L_ANCHOR or R_ANCHOR inside "\(" and "\)" (which
 120285 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the
 120286 application, as noted in XBD [Section 9.3.8](#) (on page 188). Implementations are permitted to
 120287 extend the language to interpret '^' and '\$' as anchors in these locations, and as such,
 120288 conforming applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)"
 120289 that might be interpreted as anchors.
- 120290 The ERE grammar does not permit several constructs that XBD [Section 9.4.2](#) (on page 188) and
 120291 [Section 9.4.3](#) (on page 189) specify as having undefined results:
- 120292 ORD_CHAR preceded by <backslash>
- 120293 *ERE_dupl_symbol*(s) appearing first in an ERE, or immediately following '|', '^', or '('
- 120294 '{' not part of a valid *ERE_dupl_symbol*

120295 ' | ' appearing first or last in an ERE, or immediately following ' | ' or ' (' , or
120296 immediately preceding ') ')

120297 Implementations are permitted to extend the language to allow these. Conforming applications
120298 cannot use such constructs.

120299 *A.9.5.1 BRE/ERE Grammar Lexical Conventions*

120300 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0052 [554] is applied.

120301 *A.9.5.2 RE and Bracket Expression Grammar*

120302 The removal of the *Back_open_paren Back_close_paren* option from the *nondupl_RE* specification is
120303 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2:1993 standard.
120304 Although the grammar required support for null subexpressions, this section does not describe
120305 the meaning of, and historical practice did not support, this construct.

120306 *A.9.5.3 ERE Grammar*

120307 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0052 [554] and XBD/TC2-2008/0053
120308 [916] are applied.

120309 **A.10 Directory Structure and Devices**

120310 **A.10.1 Directory Structure and Files**

120311 A description of the historical */usr/tmp* was omitted, removing any concept of differences in
120312 emphasis between the */* and */usr* directories. The descriptions of */bin*, */usr/bin*, */lib*, and */usr/lib*
120313 were omitted because they are not useful for applications. In an early draft, a distinction was
120314 made between system and application directory usage, but this was not found to be useful.

120315 The directories */* and */dev* are included because the notion of a hierarchical directory structure is
120316 key to other information presented elsewhere in POSIX.1-2017. In early drafts, it was argued that
120317 special devices and temporary files could conceivably be handled without a directory structure
120318 on some implementations. For example, the system could treat the characters *"/tmp"* as a
120319 special token that would store files using some non-POSIX file system structure. This notion was
120320 rejected by the standard developers, who required that all the files in this section be
120321 implemented via POSIX file systems.

120322 The */tmp* directory is retained in POSIX.1-2017 to accommodate historical applications that
120323 assume its availability. Implementations are encouraged to provide suitable directory names in
120324 the environment variable *TMPDIR* and applications are encouraged to use the contents of
120325 *TMPDIR* for creating temporary files.

120326 The standard files */dev/null* and */dev/tty* are required to be both readable and writable to allow
120327 applications to have the intended historical access to these files.

120328 The standard file */dev/console* has been added for alignment with the Single UNIX
120329 Specification.

120330 **A.10.2 Output Devices and Terminal Types**

120331 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/17 is applied, making it clear that the
 120332 requirements for documenting terminal support are in the system documentation.

120333 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0054 [967] is applied.

120334 **A.11 General Terminal Interface**

120335 If the implementation does not support this interface on any device types, it should behave as if
 120336 it were being used on a device that is not a terminal device (in most cases *errno* will be set to
 120337 [ENOTTY] on return from functions defined by this interface). This is based on the fact that
 120338 many applications are written to run both interactively and in some non-interactive mode, and
 120339 they adapt themselves at runtime. Requiring that they all be modified to test an environment
 120340 variable to determine whether they should try to adapt is unnecessary. On a system that
 120341 provides no general terminal interface, providing all the entry points as stubs that return
 120342 [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the
 120343 application.

120344 Although the needs of both interface implementors and application developers were addressed
 120345 throughout POSIX.1-2017, this section pays more attention to the needs of the latter. This is
 120346 because, while many aspects of the programming interface can be hidden from the user by the
 120347 application developer, the terminal interface is usually a large part of the user interface.
 120348 Although to some extent the application developer can build missing features or work around
 120349 inappropriate ones, the difficulties of doing that are greater in the terminal interface than
 120350 elsewhere. For example, efficiency prohibits the average program from interpreting every
 120351 character passing through it in order to simulate character erase, line kill, and so on. These
 120352 functions should usually be done by the operating system, possibly at the interrupt level.

120353 The *tc**() functions were introduced as a way of avoiding the problems inherent in the
 120354 traditional *ioctl*() function and in variants of it that were proposed. For example, *tcsetattr*()
 120355 is specified in place of the use of the TCSETA *ioctl*() command function. This allows specification
 120356 of all the arguments in a manner consistent with the ISO C standard unlike the varying third
 120357 argument of *ioctl*(), which is sometimes a pointer (to any of many different types) and
 120358 sometimes an **int**.

120359 The advantages of this new method include:

- 120360 It allows strict type checking.
- 120361 The direction of transfer of control data is explicit.
- 120362 Portable capabilities are clearly identified.
- 120363 The need for a general interface routine is avoided.
- 120364 Size of the argument is well-defined (there is only one type).

120365 The disadvantages include:

- 120366 No historical implementation used the new method.
- 120367 There are many small routines instead of one general-purpose one.
- 120368 The historical parallel with *fcntl*() is broken.

120369 The issue of modem control was excluded from POSIX.1-2017 on the grounds that:

- 120370 It was concerned with setting and control of hardware timers.
- 120371 The appropriate timers and settings vary widely internationally.
- 120372 Feedback from European computer manufacturers indicated that this facility was not
120373 consistent with European needs and that specification of such a facility was not a
120374 requirement for portability.

120375 A.11.1 Interface Characteristics

120376 A.11.1.1 Opening a Terminal Device File

120377 The `O_TTY_INIT` flag for `open()` has been added to POSIX.1-2017 to solve a problem
120378 encountered by applications written for earlier versions of this standard which need to open a
120379 modem or similar device and initialize all of the parameter settings. Using the
120380 `tcgetattr()-modify-tcsetattr()` method mandated by the standard could result in non-conforming
120381 behavior if the device had previously been used with non-conforming parameter settings, on
120382 implementations which do not reset the parameter settings in between the last close of the
120383 device by one application and the first open by another application. To avoid this problem, some
120384 application developers were resorting to using `memset()` to zero the **termios** structure before
120385 setting all of the standard parameters, but this risks non-conforming behavior on systems where
120386 some non-standard parameter needs a non-zero value in order for the terminal to behave in a
120387 conforming manner.

120388 On systems which do reset the parameter settings to defaults between uses of a terminal device,
120389 it is expected that either `O_TTY_INIT` will have the value zero or `open(ttypath,
120390 O_RDWR|O_TTY_INIT)` will do nothing additional.

120391 The standard developers considered an alternative solution of a special *fildev* argument for the
120392 `tcgetattr()` call to obtain default parameters. However, this would not be adequate if a system
120393 supports several different types of terminal device and the default settings need to differ
120394 between the different types. With the `O_TTY_INIT` open flag, the implementor can determine
120395 which device type is being opened.

120396 The standard developers also considered a special `POSIX_TTY_INIT` value for the **termios**
120397 structure used in `tcsetattr()`, which would reset the values if used immediately after an `open()`
120398 call. However, it was felt that this would lead to confusion amongst application developers who
120399 wanted to reset the parameters at other points, and implementations might diverge.

120400 A.11.1.2 Process Groups

120401 There is a potential race when the members of the foreground process group on a terminal leave
120402 that process group, either by exit or by changing process groups. After the last process exits the
120403 process group, but before the foreground process group ID of the terminal is changed (usually
120404 by a job control shell), it would be possible for a new process to be created with its process ID
120405 equal to the terminal's foreground process group ID. That process might then become the
120406 process group leader and accidentally be placed into the foreground on a terminal that was not
120407 necessarily its controlling terminal. As a result of this problem, the controlling terminal is
120408 defined to not have a foreground process group during this time.

120409 The cases where a controlling terminal has no foreground process group occur when all
120410 processes in the foreground process group either terminate and are waited for or join other
120411 process groups via `setpgid()` or `setsid()`. If the process group leader terminates, this is the first

120412 case described; if it leaves the process group via *setpgid()*, this is the second case described (a
120413 process group leader cannot successfully call *setsid()*). When one of those cases causes a
120414 controlling terminal to have no foreground process group, it has two visible effects on
120415 applications. The first is the value returned by *tcgetpgrp()*. The second (which occurs only in the
120416 case where the process group leader terminates) is the sending of signals in response to special
120417 input characters. The intent of POSIX.1-2017 is that no process group be wrongly identified as
120418 the foreground process group by *tcgetpgrp()* or unintentionally receive signals because of
120419 placement into the foreground.

120420 In 4.3 BSD, the old process group ID continues to be used to identify the foreground process
120421 group and is returned by the function equivalent to *tcgetpgrp()*. In that implementation it is
120422 possible for a newly created process to be assigned the same value as a process ID and then form
120423 a new process group with the same value as a process group ID. The result is that the new
120424 process group would receive signals from this terminal for no apparent reason, and
120425 POSIX.1-2017 precludes this by forbidding a process group from entering the foreground in this
120426 way. It would be more direct to place part of the requirement made by the last sentence under
120427 *fork()*, but there is no convenient way for that section to refer to the value that *tcgetpgrp()*
120428 returns, since in this case there is no process group and thus no process group ID.

120429 One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to
120430 prevent this reuse of the ID, probably in the implementation of *fork()*, as long as it is in use by
120431 the terminal.

120432 Another possibility is to recognize when the last process stops using the terminal's foreground
120433 process group ID, which is when the process group lifetime ends, and to change the terminal's
120434 foreground process group ID to a reserved value that is never used as a process ID or process
120435 group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID
120436 can then be reserved until the terminal has another foreground process group.

120437 The 4.3 BSD implementation permits the leader (and only member) of the foreground process
120438 group to leave the process group by calling the equivalent of *setpgid()* and to later return,
120439 expecting to return to the foreground. There are no known application needs for this behavior,
120440 and POSIX.1-2017 neither requires nor forbids it (except that it is forbidden for session leaders)
120441 by leaving it unspecified.

120442 A.11.1.3 The Controlling Terminal

120443 POSIX.1-2017 does not specify a mechanism by which to allocate a controlling terminal. This is
120444 normally done by a system utility (such as *getty*) and is considered an administrative feature
120445 outside the scope of POSIX.1-2017.

120446 Historical implementations allocate controlling terminals on certain *open()* calls. Since *open()* is
120447 part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required
120448 because it is not very straightforward or flexible for either implementations or applications.
120449 However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a
120450 mechanism was standardized to ensure portable, predictable behavior in *open()*.

120451 Some historical implementations deallocate a controlling terminal on the last system-wide close.
120452 This behavior in neither required nor prohibited. Even on implementations that do provide this
120453 behavior, applications generally cannot depend on it due to its system-wide nature.

120454 A.11.1.4 Terminal Access Control

120455 The access controls described in this section apply only to a process that is accessing its
120456 controlling terminal. A process accessing a terminal that is not its controlling terminal is
120457 effectively treated the same as a member of the foreground process group. While this may seem
120458 unintuitive, note that these controls are for the purpose of job control, not security, and job
120459 control relates only to the controlling terminal of a process. Normal file access permissions
120460 handle security.

120461 If the process calling *read()* or *write()* is in a background process group that is orphaned, it is not
120462 desirable to stop the process group, as it is no longer under the control of a job control shell that
120463 could put it into the foreground again. Accordingly, calls to *read()* or *write()* functions by such
120464 processes receive an immediate error return. This is different from 4.2 BSD, which kills orphaned
120465 processes that receive terminal stop signals.

120466 The foreground/background/orphaned process group check performed by the terminal driver
120467 must be repeatedly performed until the calling process moves into the foreground or until the
120468 process group of the calling process becomes orphaned. That is, when the terminal driver
120469 determines that the calling process is in the background and should receive a job control signal,
120470 it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of
120471 the calling process and then it allows the calling process to immediately receive the signal. The
120472 latter is typically performed by blocking the process so that the signal is immediately noticed.
120473 Note, however, that after the process finishes receiving the signal and control is returned to the
120474 driver, the terminal driver must re-execute the foreground/background/orphaned process
120475 group check. The process may still be in the background, either because it was continued in the
120476 background by a job control shell, or because it caught the signal and did nothing.

120477 The terminal driver repeatedly performs the foreground/background/orphaned process group
120478 checks whenever a process is about to access the terminal. In the case of *write()* or the control
120479 *tc*()* functions, the check is performed at the entry of the function. In the case of *read()*, the
120480 check is performed not only at the entry of the function, but also after blocking the process to
120481 wait for input characters (if necessary). That is, once the driver has determined that the process
120482 calling the *read()* function is in the foreground, it attempts to retrieve characters from the input
120483 queue. If the queue is empty, it blocks the process waiting for characters. When characters are
120484 available and control is returned to the driver, the terminal driver must return to the repeated
120485 foreground/background/orphaned process group check again. The process may have moved
120486 from the foreground to the background while it was blocked waiting for input characters.

120487 A.11.1.5 Input Processing and Reading Data

120488 There is no additional rationale provided for this section.

120489 A.11.1.6 Canonical Mode Input Processing

120490 The term “character” is intended here. ERASE should erase the last character, not the last byte.
120491 In the case of multi-byte characters, these two may be different.

120492 4.3 BSD has a WERASE character that erases the last “word” typed (but not any preceding
120493 <blank> or <tab> characters). A word is defined as a sequence of non-<blank> characters, with
120494 <tab> characters counted as <blank> characters. Like ERASE, WERASE does not erase beyond
120495 the beginning of the line. This WERASE feature has not been specified in POSIX.1 because it is
120496 difficult to define in the international environment. It is only useful for languages where words
120497 are delimited by <blank> characters. In some ideographic languages, such as Japanese and
120498 Chinese, words are not delimited at all. The WERASE character should presumably go back to
120499 the beginning of a sentence in those cases; practically, this means it would not be used much for

120500 those languages.

120501 It should be noted that there is a possible inherent deadlock if the application and
120502 implementation conflict on the value of {MAX_CANON}. With ICANON set (if IXOFF is
120503 enabled) and more than {MAX_CANON} characters transmitted without a <linefeed>,
120504 transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never
120505 arrive, and the *read()* will never be satisfied.

120506 An application should not set IXOFF if it is using canonical mode unless it knows that (even in
120507 the face of a transmission error) the conditions described previously cannot be met or unless it is
120508 prepared to deal with the possible deadlock in some other way, such as timeouts.

120509 It should also be noted that this can be made to happen in non-canonical mode if the trigger
120510 value for sending IXOFF is less than VMIN and VTIME is zero.

120511 A.11.1.7 Non-Canonical Mode Input Processing

120512 Some points to note about MIN and TIME:

120513 1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and
120514 TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0,
120515 both MIN and TIME play a role in that MIN is satisfied with the receipt of a single
120516 character.

120517 2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer,
120518 while in case C (MIN=0, TIME>0), TIME represents a read timer.

120519 These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where
120520 MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a
120521 program would like to process at least MIN characters at a time. In case A, the inter-character
120522 timer is activated by a user as a safety measure; in case B, it is turned off.

120523 Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable
120524 to screen-based applications that need to know if a character is present in the input queue before
120525 refreshing the screen. In case C, the read is timed; in case D, it is not.

120526 Another important note is that MIN is always just a minimum. It does not denote a record
120527 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20
120528 characters are returned to the user. In the special case of MIN=0, this still applies: if more than
120529 one character is available, they all will be returned immediately.

120530 A.11.1.8 Writing Data and Output Processing

120531 There is no additional rationale provided for this section.

120532 A.11.1.9 Special Characters

120533 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0055 [745] is applied.

120534 A.11.1.10 Modem Disconnect

120535 There is no additional rationale provided for this section.

120536 A.11.1.11 *Closing a Terminal Device File*

120537 POSIX.1-2017 does not specify that a *close()* on a terminal device file include the equivalent of a
120538 call to *tcflow(fd,TCOON)*.

120539 An implementation that discards output at the time *close()* is called after reporting the return
120540 value to the *write()* call that data was written does not conform with POSIX.1-2017. An
120541 application has functions such as *tcdrain()*, *tcflush()*, and *tcflow()* available to obtain the detailed
120542 behavior it requires with respect to flushing of output.

120543 At the time of the last close on a terminal device, an application relinquishes any ability to exert
120544 flow control via *tcflow()*.

120545 **A.11.2 Parameters that Can be Set**120546 A.11.2.1 *The termios Structure*

120547 This structure is part of an interface that, in general, retains the historic grouping of flags.
120548 Although a more optimal structure for implementations may be possible, the degree of change
120549 to applications would be significantly larger.

120550 A.11.2.2 *Input Modes*

120551 Some historical implementations treated a long break as multiple events, as many as one per
120552 character time. The wording in POSIX.1 explicitly prohibits this.

120553 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,
120554 it is historically supported. Therefore, applications may be using ISTRIP, and there is no
120555 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit
120556 input bytes and may not be connected directly to the hardware terminal device (for example,
120557 when a connection traverses a network).

120558 Also, there is no requirement in general that the terminal device ensures that high-order bits
120559 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit
120560 characters, which are common.

120561 In dealing with multi-byte characters, the consequences of a parity error in such a character, or
120562 in an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and
120563 are best dealt with by the application processing the multi-byte characters.

120564 A.11.2.3 *Output Modes*

120565 POSIX.1 does not describe post-processing of output to a terminal or detailed control of that
120566 from a conforming application. (That is, translation of <newline> to <carriage-return> followed
120567 by <linefeed> or <tab> processing.) There is nothing that a conforming application should do to
120568 its output for a terminal because that would require knowledge of the operation of the terminal.
120569 It is the responsibility of the operating system to provide post-processing appropriate to the
120570 output device, whether it is a terminal or some other type of device.

120571 Extensions to POSIX.1 to control the type of post-processing already exist and are expected to
120572 continue into the future. The control of these features is primarily to adjust the interface between
120573 the system and the terminal device so the output appears on the display correctly. This should
120574 be set up before use by any application.

120575 In general, both the input and output modes should not be set absolutely, but rather modified
120576 from the inherited state.

120577 A.11.2.4 Control Modes

120578 This section could be misread that the symbol ``CSIZE'' is a title in the **termios** *c_flag* field.
120579 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1
120580 (and the caveats about typography) would indicate.

120581 A.11.2.5 Local Modes

120582 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still
120583 allowing single-character input.

120584 The ECHONL function historically has been in many implementations. Since there seems to be
120585 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

120586 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is
120587 permitted as a compromise depending on what the actual terminal hardware can do. Erasing
120588 characters and lines is preferred, but is not always possible.

120589 A.11.2.6 Special Control Characters

120590 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical
120591 implementations. Only when backwards-compatibility of object code is a serious concern to an
120592 implementor should an implementation continue this practice. Correct applications that work
120593 with the overlap (at the source level) should also work if it is not present, but not the reverse.

120594 A.12 Utility Conventions

120595 A.12.1 Utility Argument Syntax

120596 The standard developers considered that recent trends toward diluting the SYNOPSIS sections
120597 of historical reference pages to the equivalent of:

120598 `command [options] [operands]`

120599 were a disservice to the reader. Therefore, considerable effort was placed into rigorous
120600 definitions of all the command line arguments and their interrelationships. The relationships
120601 depicted in the synopses are normative parts of POSIX.1-2017; this information is sometimes
120602 repeated in textual form, but that is only for clarity within context.

120603 The use of ``undefined'' for conflicting argument usage and for repeated usage of the same
120604 option is meant to prevent conforming applications from using conflicting arguments or
120605 repeated options unless specifically allowed (as is the case with *ls*, which allows simultaneous,
120606 repeated use of the *-C*, *-l*, and *-1* options). Many historical implementations will tolerate this
120607 usage, choosing either the first or the last applicable argument. This tolerance can continue, but
120608 conforming applications cannot rely upon it. (Other implementations may choose to print usage
120609 messages instead.)

120610 The use of ``undefined'' for conflicting argument usage also allows an implementation to make

120611 reasonable extensions to utilities where the implementor considers mutually-exclusive options
120612 according to POSIX.1-2017 to have a sensible meaning and result.

120613 POSIX.1-2017 does not define the result of a command when an option-argument or operand is
120614 not followed by ellipses and the application specifies more than one of that option-argument or
120615 operand. This allows an implementation to define valid (although non-standard) behavior for
120616 the utility when more than one such option or operand is specified.

120617 The requirements for option-arguments are summarized as follows:

	SYNOPSIS Shows:	
	<i>-a arg</i>	<i>-c[arg]</i>
Conforming application uses:	<i>-a arg</i>	<i>-carg or -c</i>
System supports:	<i>-a arg</i> and <i>-aarg</i>	<i>-carg</i> and <i>-c</i>
Non-conforming applications may use:	<i>-aarg</i>	N/A

120623 Earlier versions of this standard included obsolescent syntax which showed some options with
120624 (mandatory) adjacent option-arguments in the SYNOPSIS for some utilities. These have since
120625 been removed. For all options with mandatory option-arguments, the SYNOPSIS now shows
120626 <blank> characters between the option and the option-argument; however, historical usage has
120627 not been consistent in this area; therefore, <blank> characters are required to be used by
120628 conforming applications and to be handled by all implementations, but implementations are
120629 also required to handle an adjacent option-argument in order to preserve backwards-
120630 compatibility for old scripts. One of the justifications for selecting the multiple-argument
120631 method was that the single-argument case is inherently ambiguous when the option-argument
120632 can legitimately be a null string.

120633 POSIX.1-2017 explicitly states that digits are permitted as operands and option-arguments. The
120634 lower and upper bounds for the values of the numbers used for operands and option-arguments
120635 were derived from the ISO C standard values for {LONG_MIN} and {LONG_MAX}. The
120636 requirement on the standard utilities is that numbers in the specified range do not cause a
120637 syntax error, although the specification of a number need not be semantically correct for a
120638 particular operand or option-argument of a utility. For example, the specification of:

120639 `dd obs=3000000000`

120640 would yield undefined behavior for the application and could be a syntax error because the
120641 number 3 000 000 000 is outside of the range -2 147 483 647 to +2 147 483 647. On the other hand:

120642 `dd obs=2000000000`

120643 may cause some error, such as “blocksize too large”, rather than a syntax error.

120644 POSIX.1-2008, Technical Corrigendum 2, XBD/TC2-2008/0056 [584] and XBD/TC2-2008/0057
120645 [813] are applied.

120646 A.12.2 Utility Syntax Guidelines

120647 This section is based on the rules listed in the SVID. It was included for two reasons:

- 120648 1. The individual utility descriptions in XCU [Chapter 4](#) (on page 2453) needed a set of
120649 common (although not universal) actions on which they could anchor their descriptions
120650 of option and operand syntax. Most of the standard utilities actually do use these
120651 guidelines, and many of their historical implementations use the *getopt()* function for
120652 their parsing. Therefore, it was simpler to cite the rules and merely identify exceptions.

120653 2. Developers of conforming applications need suggested guidelines if the POSIX
120654 community is to avoid the chaos of historical UNIX system command syntax.

120655 It is recommended that all *future* utilities and applications use these guidelines to enhance “user
120656 portability”. The fact that some historical utilities could not be changed (to avoid breaking
120657 historical applications) should not deter this future goal.

120658 The voluntary nature of the guidelines is highlighted by repeated uses of the word *should*
120659 throughout. This usage should not be misinterpreted to imply that utilities that claim
120660 conformance in their OPTIONS sections do not always conform.

120661 Guidelines 1 and 2 encourage utility writers to use only characters from the portable character
120662 set because use of locale-specific characters may make the utility inaccessible from other locales.
120663 Use of uppercase letters is discouraged due to problems associated with porting utilities to
120664 systems that do not distinguish between uppercase and lowercase characters in filenames. Use
120665 of non-alphanumeric characters is discouraged due to the number of utilities that treat non-
120666 alphanumeric characters in “special” ways depending on context (such as the shell using white-
120667 space characters to delimit arguments, various quote characters for quoting, the <dollar-sign> to
120668 introduce variable expansion, etc.).

120669 In XCU [Section 2.9.1](#) (on page 2365), it is further stated that a command used in the Shell
120670 Command Language cannot be named with a trailing <colon>.

120671 Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the
120672 character set to allow compatibility with historical usage. Historical practice allows the use of
120673 digits wherever practical, and there are no portability issues that would prohibit the use of
120674 digits. In fact, from an internationalization viewpoint, digits (being non-language-dependent)
120675 are preferable over letters (a `-2` is intuitively self-explanatory to any user, while in the `-f filename`
120676 the letter ‘f’ is a mnemonic aid only to speakers of Latin-based languages where “filename”
120677 happens to translate to a word that begins with ‘f’). Since Guideline 3 still retains the word
120678 “single”, multi-digit options are not allowed. Instances of historical utilities that used them have
120679 been marked obsolescent, with the numbers being changed from option names to option-
120680 arguments.

120681 It was difficult to achieve a satisfactory solution to the problem of name space in option
120682 characters. When the standard developers desired to extend the historical `cc` utility to accept
120683 ISO C standard programs, they found that all of the portable alphabet was already in use by
120684 various vendors. Thus, they had to devise a new name, `c89` (now superseded by `c99`), rather than
120685 something like `cc -X`. There were suggestions that implementors be restricted to providing
120686 extensions through various means (such as using a <plus-sign> as the option delimiter or using
120687 option characters outside the alphanumeric set) that would reserve all of the remaining
120688 alphanumeric characters for future POSIX standards. These approaches were resisted because
120689 they lacked the historical style of UNIX systems. Furthermore, if a vendor-provided option
120690 should become commonly used in the industry, it would be a candidate for standardization. It
120691 would be desirable to standardize such a feature using historical practice for the syntax (the
120692 semantics can be standardized with any syntax). This would not be possible if the syntax was
120693 one reserved for the vendor. However, since the standardization process may lead to minor
120694 changes in the semantics, it may prove to be better for a vendor to use a syntax that will not be
120695 affected by standardization.

120696 Guideline 8 includes the concept of <comma>-separated lists in a single argument. It is up to the
120697 utility to parse such a list itself because `getopt()` just returns the single string. This situation was
120698 retained so that certain historical utilities would not violate the guidelines. Applications
120699 preparing for international use should be aware of an occasional problem with
120700 <comma>-separated lists: in some locales, the <comma> is used as the radix character. Thus, if
120701 an application is preparing operands for a utility that expects a <comma>-separated list, it

120702 should avoid generating non-integer values through one of the means that is influenced by
120703 setting the `LC_NUMERIC` variable (such as `awk`, `bc`, `printf`, or `printf()`).

120704 Unless explicitly stated otherwise in the utility description, Guideline 9 requires applications to
120705 put options before operands, and requires utilities to accept any such usage without
120706 misinterpreting operands as options. For example, if an implementation of the `printf` utility
120707 supports a `-e` option as an extension, the command:

```
120708 printf %s -e
```

120709 must output the string `"-e"` without interpreting the `-e` as an option. Similarly, the command:

```
120710 ls myfile -l
```

120711 must interpret the `-l` argument as a second file operand, not as a `-l` option.

120712 Applications calling any utility with a first operand starting with `'-'` should usually specify `--`,
120713 as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS
120714 in the Shell and Utilities volume of POSIX.1-2017 does not specify any options; implementations
120715 may provide options as extensions to the Shell and Utilities volume of POSIX.1-2017. The
120716 standard utilities that do not support Guideline 10 indicate that fact in the OPTIONS section of
120717 the utility description.

120718 Guideline 7 allows any string to be an option-argument; an option-argument can begin with any
120719 character, can be `-` or `--`, and can be an empty string. For example, the commands `pr -h -`, `pr -h`
120720 `--`, `pr -h -d`, `pr -h +2`, and `pr -h ''` contain the option-arguments `-`, `--`, `-d`, `+2`, and an empty
120721 string, respectively. Conversely, the command `pr -h -- -d` treats `-d` as an option, not as an
120722 argument, because the `--` is an option-argument here, not a delimiter.

120723 Guideline 11 was modified to clarify that the order of different options should not matter
120724 relative to one another. However, the order of repeated options that also have option-arguments
120725 may be significant; therefore, such options are required to be interpreted in the order that they
120726 are specified. The `make` utility is an instance of a historical utility that uses repeated options in
120727 which the order is significant. Multiple files are specified by giving multiple instances of the `-f`
120728 option; for example:

```
120729 make -f common_header -f specific_rules target
```

120730 Guideline 13 does not imply that all of the standard utilities automatically accept the operand
120731 `'-'` to mean standard input or output, nor does it specify the actions of the utility upon
120732 encountering multiple `'-'` operands. It simply says that, by default, `'-'` operands are not used
120733 for other purposes in the file reading or writing (but not when using `stat()`, `unlink()`, `touch`, and
120734 so on) utilities. In earlier versions of this standard, all information concerning actual treatment of
120735 the `'-'` operand is found in the individual utility sections. Many implementations, however,
120736 treated `'-'` as standard input or output and many applications depended on this behavior even
120737 though it was not standard. This behavior is now implementation-defined. Portable applications
120738 should not use `'-'` to mean standard input or output unless it is explicitly stated to do so in the
120739 utility description and they should always use `'./-'` if they intend to refer to a file named `-` in
120740 the current working directory.

120741 Guideline 14 is intended to prohibit implementations that would treat the command `ls -l -d` as if
120742 it were `ls -- -l -d` or `ls -l -- -d`.

120743 The standard permits implementations to have extensions that violate the Utility Syntax
120744 Guidelines so long as when the utility is used in line with the forms defined by the standard it
120745 follows the Utility Syntax Guidelines. Thus, `head-42file` and `ls--help` are permitted extensions.
120746 The intent is to allow extensions so long as the standard form is accepted and follows the
120747 guidelines.

120748 An area of concern was that as implementations mature, implementation-defined utilities and
 120749 implementation-defined utility options will result. The idea was expressed that there needed to
 120750 be a standard way, say an environment variable or some such mechanism, to identify
 120751 implementation-defined utilities separately from standard utilities that may have the same
 120752 name. It was decided that there already exist several ways of dealing with this situation and that
 120753 it is outside of the scope to attempt to standardize in the area of non-standard items. A method
 120754 that exists on some historical implementations is the use of the so-called **/local/bin** or
 120755 **/usr/local/bin** directory to separate local or additional copies or versions of utilities. Another
 120756 method that is also used is to isolate utilities into completely separate domains. Still another
 120757 method to ensure that the desired utility is being used is to request the utility by its full
 120758 pathname. There are many approaches to this situation; the examples given above serve to
 120759 illustrate that there is more than one.

120760 **A.13 Headers**

120761 **A.13.1 Format of Entries**

120762 Each header reference page has a common layout of sections describing the interface. This
 120763 layout is similar to the manual page or “man” page format shipped with most UNIX systems,
 120764 and each header has sections describing the SYNOPSIS and DESCRIPTION. These are the two
 120765 sections that relate to conformance.

120766 Additional sections are informative, and add considerable information for the application
 120767 developer. APPLICATION USAGE sections provide additional caveats, issues, and
 120768 recommendations to the developer. RATIONALE sections give additional information on the
 120769 decisions made in defining the interface.

120770 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in
 120771 the future, and often cautions the developer to architect the code to account for a change in this
 120772 area. Note that a future directions statement should not be taken as a commitment to adopt a
 120773 feature or interface in the future.

120774 The CHANGE HISTORY section describes when the interface was introduced, and how it has
 120775 changed.

120776 Option labels and margin markings in the page can be useful in guiding the application
 120777 developer.


120778 **A.13.2 Removed Headers in Issue 7**

120779 The headers removed in Issue 7 (from the Issue 6 base document) are as follows:

120780
 120781

Removed Headers in Issue 7	
<sys/timeb.h>	<ucontext.h>

120782

 *Rationale (Informative)*

120783

Part B:

120784


System Interfaces

120785

The Open Group

120786

The Institute of Electrical and Electronics Engineers, Inc.



Rationale for System Interfaces

120789 **B.1 Introduction**

120790 **B.1.1 Change History**

120791 The change history is provided as an informative section, to track changes from earlier versions
120792 of this standard.

120793 The following sections describe changes made to the System Interfaces volume of POSIX.1-2017
120794 since Issue 6 of the base document. The CHANGE HISTORY section for each entry details the
120795 technical changes that have been made to that entry from Issue 5. Changes between earlier
120796 versions of the base document and Issue 5 are not included.

120797 **Changes from Issue 6 to Issue 7 (POSIX.1-2008)**

120798 The following list summarizes the major changes that were made in the System Interfaces
120799 volume of POSIX.1-2017 from Issue 6 to Issue 7:

120800 The Open Group Technical Standard, 2006, Extended API Set Part 1 is incorporated.

120801 The Open Group Technical Standard, 2006, Extended API Set Part 2 is incorporated.

120802 The Open Group Technical Standard, 2006, Extended API Set Part 3 is incorporated.

120803 The Open Group Technical Standard, 2006, Extended API Set Part 4 is incorporated.

120804 Existing functionality is aligned with ISO/IEC 9899:1999, Programming Languages — C,
120805 ISO/IEC 9899:1999/Cor.2:2004(E)

120806 Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses
120807 to ISO/IEC defect reports against ISO/IEC 9945 are applied.

120808 The Open Group corrigenda and resolutions are applied.

120809 Features, marked legacy or obsolescent in the base document, have been considered for
120810 removal in this version.

120811 The options within the standard have been revised.

120812 **New Features in Issue 7**

120813 The functions first introduced in Issue 7 (over the Issue 6 base document) are as follows:

120814

120815

120816

120817

120818

120819

120820

120821

120822

120823

120824

120825

120826

120827

120828

120829

120830

120831

120832

120833

120834

120835

120836

120837

120838

120839

120840

120841

120842

120843

120844

120845

New Functions in Issue 7

<i>alphasort()</i>	<i>iswdigit_l()</i>	<i>strfmon_l()</i>
<i>dirfd()</i>	<i>iswgraph_l()</i>	<i>strncasemp_l()</i>
<i>dprintf()</i>	<i>iswlower_l()</i>	<i>strndup()</i>
<i>duplocale()</i>	<i>iswprint_l()</i>	<i>strlen()</i>
<i>faccessat()</i>	<i>iswpunct_l()</i>	<i>strsignal()</i>
<i>fchmodat()</i>	<i>iswspace_l()</i>	<i>strxfrm_l()</i>
<i>fchownat()</i>	<i>iswupper_l()</i>	<i>symlinkat()</i>
<i>fdopendir()</i>	<i>iswxdigit_l()</i>	<i>tolower_l()</i>
<i>fexecve()</i>	<i>isxdigit_l()</i>	<i>toupper_l()</i>
<i>fnmopen()</i>	<i>linkat()</i>	<i>towctrans_l()</i>
<i>freelocale()</i>	<i>mbsnrtowcs()</i>	<i>towlower_l()</i>
<i>fstatat()</i>	<i>mkdirat()</i>	<i>towupper_l()</i>
<i>futimens()</i>	<i>mkdtemp()</i>	<i>unlinkat()</i>
<i>getdelim()</i>	<i>mkfifoat()</i>	<i>uselocale()</i>
<i>getline()</i>	<i>mknodat()</i>	<i>utimensat()</i>
<i>isalnum_l()</i>	<i>newlocale()</i>	<i>vdprintf()</i>
<i>isalpha_l()</i>	<i>openat()</i>	<i>wcpcpy()</i>
<i>isblank_l()</i>	<i>open_memstream()</i>	<i>wcpncpy()</i>
<i>isctrl_l()</i>	<i>open_wmemstream()</i>	<i>wcscasemp_l()</i>
<i>isdigit_l()</i>	<i>psiginfo()</i>	<i>wcscasemp_l()</i>
<i>isgraph_l()</i>	<i>psignal()</i>	<i>wcscoll_l()</i>
<i>islower_l()</i>	<i>pthread_mutexattr_getrobust()</i>	<i>wcsdup()</i>
<i>isprint_l()</i>	<i>pthread_mutexattr_setrobust()</i>	<i>wcsncasemp_l()</i>
<i>ispunct_l()</i>	<i>pthread_mutex_consistent()</i>	<i>wcsncasemp_l()</i>
<i>isspace_l()</i>	<i>readlinkat()</i>	<i>wcsnlen()</i>
<i>isupper_l()</i>	<i>renameat()</i>	<i>wcsnrtombs()</i>
<i>iswalnum_l()</i>	<i>scandir()</i>	<i>wcsxfrm_l()</i>
<i>iswalphal_l()</i>	<i>stpncpy()</i>	<i>wctrans_l()</i>
<i>iswblank_l()</i>	<i>stpncpy()</i>	<i>wctype_l()</i>
<i>iswcntrl_l()</i>	<i>strcasemp_l()</i>	
<i>iswctype_l()</i>	<i>strcoll_l()</i>	

120846

Newly Mandated Functions in Issue 7

120847

120848 The functions that were previously part of an option group but are now mandatory in Issue 7 are as follows:

	Newly Mandated Functions in Issue 7		
120849			
120850	<i>aio_cancel()</i>	<i>pthread_atfork()</i>	<i>pthread_rwlock_tryrdlock()</i>
120851	<i>aio_error()</i>	<i>pthread_attr_destroy()</i>	<i>pthread_rwlock_trywrlock()</i>
120852	<i>aio_fsync()</i>	<i>pthread_attr_getdetachstate()</i>	<i>pthread_rwlock_unlock()</i>
120853	<i>aio_read()</i>	<i>pthread_attr_getguardsize()</i>	<i>pthread_rwlock_wrlock()</i>
120854	<i>aio_return()</i>	<i>pthread_attr_getschedparam()</i>	<i>pthread_rwlockattr_destroy()</i>
120855	<i>aio_suspend()</i>	<i>pthread_attr_init()</i>	<i>pthread_rwlockattr_init()</i>
120856	<i>aio_write()</i>	<i>pthread_attr_setdetachstate()</i>	<i>pthread_self()</i>
120857	<i>asctime_r()</i>	<i>pthread_attr_setguardsize()</i>	<i>pthread_setcancelstate()</i>
120858	<i>catclose()</i>	<i>pthread_attr_setschedparam()</i>	<i>pthread_setcanceltype()</i>
120859	<i>catgets()</i>	<i>pthread_barrier_destroy()</i>	<i>pthread_setspecific()</i>
120860	<i>catopen()</i>	<i>pthread_barrier_init()</i>	<i>pthread_spin_destroy()</i>
120861	<i>clock_getres()</i>	<i>pthread_barrier_wait()</i>	<i>pthread_spin_init()</i>
120862	<i>clock_gettime()</i>	<i>pthread_barrierattr_destroy()</i>	<i>pthread_spin_lock()</i>
120863	<i>clock_nanosleep()</i>	<i>pthread_barrierattr_init()</i>	<i>pthread_spin_trylock()</i>
120864	<i>clock_settime()</i>	<i>pthread_cancel()</i>	<i>pthread_spin_unlock()</i>
120865	<i>ctime_r()</i>	<i>pthread_cleanup_pop()</i>	<i>pthread_testcancel()</i>
120866	<i>dlclose()</i>	<i>pthread_cleanup_push()</i>	<i>putc_unlocked()</i>
120867	<i>derror()</i>	<i>pthread_cond_broadcast()</i>	<i>putchar_unlocked()</i>
120868	<i>dlopen()</i>	<i>pthread_cond_destroy()</i>	<i>pwrite()</i>
120869	<i>dlsym()</i>	<i>pthread_cond_init()</i>	<i>rand_r()</i>
120870	<i>fchdir()</i>	<i>pthread_cond_signal()</i>	<i>readdir_r()</i>
120871	<i>flockfile()</i>	<i>pthread_cond_timedwait()</i>	<i>sem_close()</i>
120872	<i>fstatvfs()</i>	<i>pthread_cond_wait()</i>	<i>sem_destroy()</i>
120873	<i>ftrylockfile()</i>	<i>pthread_condattr_destroy()</i>	<i>sem_getvalue()</i>
120874	<i>funlockfile()</i>	<i>pthread_condattr_getclock()</i>	<i>sem_init()</i>
120875	<i>getc_unlocked()</i>	<i>pthread_condattr_init()</i>	<i>sem_open()</i>
120876	<i>getchar_unlocked()</i>	<i>pthread_condattr_setclock()</i>	<i>sem_post()</i>
120877	<i>getgrgid_r()</i>	<i>pthread_create()</i>	<i>sem_timedwait()</i>
120878	<i>getgrnam_r()</i>	<i>pthread_detach()</i>	<i>sem_trywait()</i>
120879	<i>getlogin_r()</i>	<i>pthread_equal()</i>	<i>sem_unlink()</i>
120880	<i>getpgid()</i>	<i>pthread_exit()</i>	<i>sem_wait()</i>
120881	<i>getpwnam_r()</i>	<i>pthread_getspecific()</i>	<i>sigqueue()</i>
120882	<i>getpwuid_r()</i>	<i>pthread_join()</i>	<i>sigqueue()</i>
120883	<i>getsid()</i>	<i>pthread_key_create()</i>	<i>sigtimedwait()</i>
120884	<i>getsubopt()</i>	<i>pthread_key_delete()</i>	<i>sigwaitinfo()</i>
120885	<i>gmtime_r()</i>	<i>pthread_mutex_destroy()</i>	<i>statvfs()</i>
120886	<i>iconv()</i>	<i>pthread_mutex_init()</i>	<i>strcasemp()</i>
120887	<i>iconv_close()</i>	<i>pthread_mutex_lock()</i>	<i>strdup()</i>
120888	<i>iconv_open()</i>	<i>pthread_mutex_timedlock()</i>	<i>strerror_r()</i>
120889	<i>lchown()</i>	<i>pthread_mutex_trylock()</i>	<i>strfmon()</i>
120890	<i>lio_listio()</i>	<i>pthread_mutex_unlock()</i>	<i>strncasemp()</i>
120891	<i>localtime_r()</i>	<i>pthread_mutexattr_destroy()</i>	<i>strtok_r()</i>
120892	<i>mkstemp()</i>	<i>pthread_mutexattr_gettype()</i>	<i>tcgetsid()</i>
120893	<i>mmap()</i>	<i>pthread_mutexattr_init()</i>	<i>timer_create()</i>
120894	<i>mprotect()</i>	<i>pthread_mutexattr_settype()</i>	<i>timer_delete()</i>
120895	<i>munmap()</i>	<i>pthread_once()</i>	<i>timer_getoverrun()</i>
120896	<i>nanosleep()</i>	<i>pthread_rwlock_destroy()</i>	<i>timer_gettime()</i>
120897	<i>nl_langinfo()</i>	<i>pthread_rwlock_init()</i>	<i>timer_settime()</i>
120898	<i>poll()</i>	<i>pthread_rwlock_rdlock()</i>	<i>truncate()</i>
120899	<i>posix_trace_timedgetnext_event()</i>	<i>pthread_rwlock_timedrdlock()</i>	<i>ttynam_r()</i>
120900	<i>pread()</i>	<i>pthread_rwlock_timedwrlock()</i>	<i>waitid()</i>

120901 **Obsolescent Functions in Issue 7**

120902 The base functions moved to obsolescent status in Issue 7 (from the Issue 6 base document) are
 120903 as follows:

120904

120905

120906

120907

120908

Obsolescent Base Functions in Issue 7

<i>asctime()</i>	<i>gets()</i>
<i>asctime_r()</i>	<i>rand_r()</i>
<i>ctime()</i>	<i>tmpnam()</i>
<i>ctime_r()</i>	<i>utime()</i>

120909

120910

The XSI functions moved to obsolescent status in Issue 7 (from the Issue 6 base document) are as follows:

120911

120912

120913

120914

120915

120916

120917

120918

120919

Obsolescent XSI Functions in Issue 7

<i>_longjmp()</i>	<i>pthread_getconcurrency()</i>	<i>sigset()</i>
<i>_setjmp()</i>	<i>pthread_setconcurrency()</i>	<i>siginterrupt()</i>
<i>_tolower()</i>	<i>setitimer()</i>	<i>tmpnam()</i>
<i>_toupper()</i>	<i>setpggrp()</i>	<i>toascii()</i>
<i>ftw()</i>	<i>sighold()</i>	<i>ulimit()</i>
<i>getitimer()</i>	<i>sigignore()</i>	
<i>gettimeofday()</i>	<i>sigpause()</i>	
<i>isascii()</i>	<i>sigrelse()</i>	

120920

Removed Functions and Symbols in Issue 7

120921

The functions and symbols removed in Issue 7 (from the Issue 6 base document) are as follows:

120922

120923

120924

120925

120926

120927

120928

120929

120930

120931

Removed Functions and Symbols in Issue 7

<i>bcmp()</i>	<i>gethostbyaddr()</i>	<i>rindex()</i>
<i>bcopy()</i>	<i>gethostbyname()</i>	<i>scalb()</i>
<i>bsd_signal()</i>	<i>getwd()</i>	<i>setcontext()</i>
<i>bzero()</i>	<i>h_errno</i>	<i>swapcontext()</i>
<i>ecvt()</i>	<i>index()</i>	<i>ualarm()</i>
<i>fcvt()</i>	<i>makecontext()</i>	<i>usleep()</i>
<i>ftime()</i>	<i>mktemp()</i>	<i>vfork()</i>
<i>gcvt()</i>	<i>pthread_attr_getstackaddr()</i>	<i>wcswcs()</i>
<i>getcontext()</i>	<i>pthread_attr_setstackaddr()</i>	

120932 **B.1.2 Relationship to Other Formal Standards**

120933

There is no additional rationale provided for this section.

120934 **B.1.3 Format of Entries**

120935

120936

120937

120938

Each system interface reference page has a common layout of sections describing the interface. This layout is similar to the manual page or “man” page format shipped with most UNIX systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS. These are the four sections that relate to conformance.

120939

120940

Additional sections are informative, and add considerable information for the application developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections

120941 provide additional caveats, issues, and recommendations to the developer. RATIONALE
120942 sections give additional information on the decisions made in defining the interface.

120943 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in
120944 the future, and often cautions the developer to architect the code to account for a change in this
120945 area. Note that a future directions statement should not be taken as a commitment to adopt a
120946 feature or interface in the future.

120947 The CHANGE HISTORY section describes when the interface was introduced, and how it has
120948 changed.

120949 Option labels and margin markings in the page can be useful in guiding the application
120950 developer.

120951 **B.2 General Information**

120952 **B.2.1 Use and Implementation of Interfaces**

120953 The information concerning the use of functions was adapted from a description in the ISO C
120954 standard. Here is an example of how an application program can protect itself from functions
120955 that may or may not be macros, rather than true functions:

120956 The *atoi()* function may be used in any of several ways:

120957 By use of its associated header (possibly generating a macro expansion):

```
120958 #include <stdlib.h>
120959 /* ... */
120960 i = atoi(str);
```

120961 By use of its associated header (assuredly generating a true function call):

```
120962 #include <stdlib.h>
120963 #undef atoi
120964 /* ... */
120965 i = atoi(str);
```

120966 or:

```
120967 #include <stdlib.h>
120968 /* ... */
120969 i = (atoi) (str);
```

120970 By explicit declaration:

```
120971 extern int atoi (const char *);
120972 /* ... */
120973 i = atoi(str);
```

120974 By implicit declaration:

```
120975 /* ... */
120976 i = atoi(str);
```

120977 (Assuming no function prototype is in scope. This is not allowed by the ISO C standard for
120978 functions with variable arguments; furthermore, parameter type conversion “widening” is

120979 subject to different rules in this case.)

120980 Note that the ISO C standard reserves names starting with '_' for the compiler. Therefore, the
120981 compiler could, for example, implement an intrinsic, built-in function `_asm_builtin_atoi()`, which
120982 it recognized and expanded into inline assembly code. Then, in `<stdlib.h>`, there could be the
120983 following:

```
120984 #define atoi(X) _asm_builtin_atoi(X)
```

120985 The user's "normal" call to `atoi()` would then be expanded inline, but the implementor would
120986 also be required to provide a callable function named `atoi()` for use when the application
120987 requires it; for example, if its address is to be stored in a function pointer variable.

120988 Implementors should note that since applications can **#undef** a macro in order to ensure that the
120989 function is used, this means that it is not safe for implementations to use the names of any
120990 standard functions in macro values, since the application could use **#undef** to ensure that no
120991 macro exists and then use the same name for an identifier with local scope. For example,
120992 historically it was common for a `getchar()` macro to be defined in `<stdio.h>` as:

```
120993 #define getchar()getc(stdin)
```

120994 This definition does not conform, because an application is allowed to use the identifier `getc`
120995 with local scope, and the expansion of the `getchar()` macro would then pick up the local `getc`.
120996 The following is conforming code, but would not compile with the above definition of `getchar()`:

```
120997 #include <stdio.h>
```

```
120998 #undef getc
```

```
120999 int main(void)
```

```
121000 {
```

```
121001     int getc;
```

```
121002     getc = getchar();
```

```
121003     return getc;
```

```
121004 }
```

121005 This does not only affect function-like macros. For example, the following definition does not
121006 conform because there could be a local `sysconf` variable in scope when `SIGRTMIN` is expanded:

```
121007 #define SIGRTMIN ((int)sysconf(_SC_SIGRT_MIN))
```

121008 Implementors can avoid the problem by using aliases for standard functions instead of the
121009 actual function, with names that conforming applications cannot use for local variables. For
121010 example:

```
121011 #define SIGRTMIN ((int)__sysconf(_SC_SIGRT_MIN))
```

121012 B.2.2 The Compilation Environment

121013 B.2.2.1 POSIX.1 Symbols

121014 This and the following section address the issue of "name space pollution". The ISO C standard
121015 requires that the name space beyond what it reserves not be altered except by explicit action of
121016 the application developer. This section defines the actions to add the POSIX.1 symbols for those
121017 headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the

121018 XSI option extends the base standard.

121019 When headers are used to provide symbols, there is a potential for introducing symbols that the
 121020 application developer cannot predict. Ideally, each header should only contain one set of
 121021 symbols, but this is not practical for historical reasons. Thus, the concept of feature test macros is
 121022 included. Two feature test macros are explicitly defined by POSIX.1-2017; it is expected that
 121023 future versions may add to this.

121024 **Note:** Feature test macros allow an application to announce to the implementation its desire to have
 121025 certain symbols and prototypes exposed. They should not be confused with the version test
 121026 macros and constants for options in `<unistd.h>` which are the implementation's way of
 121027 announcing functionality to the application.

121028 It is further intended that these feature test macros apply only to the headers specified by
 121029 POSIX.1-2017. Implementations are expressly permitted to make visible symbols not specified
 121030 by POSIX.1-2017, within both POSIX.1 and other headers, under the control of feature test
 121031 macros that are not defined by POSIX.1-2017.

121032 **The `_POSIX_C_SOURCE` Feature Test Macro**

121033 The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been
 121034 superseded by `_POSIX_C_SOURCE`. This symbol will allow implementations to support various
 121035 versions of this standard simultaneously. For instance, when `_POSIX_C_SOURCE` is defined as
 121036 200809L, the system should make visible the same name space as permitted and required by the
 121037 POSIX.1-2017 standard. A special case is the one where the implementation wishes to make
 121038 available support for the 1990 version of the POSIX standard, in which instance when either
 121039 `_POSIX_SOURCE` is defined or `_POSIX_C_SOURCE` is defined as 1, the system should make
 121040 visible the same name space as permitted and required by the POSIX.1-1990 standard.

121041 It is expected that C bindings to future POSIX standards will define new values for
 121042 `_POSIX_C_SOURCE`, with each new value reserving the name space for that new standard.

121043 **The `_XOPEN_SOURCE` Feature Test Macro**

121044 The feature test macro `_XOPEN_SOURCE` is provided as the announcement mechanism for the
 121045 application that it requires functionality from the Single UNIX Specification. `_XOPEN_SOURCE`
 121046 must be defined to the value 700 before the inclusion of any header to enable the functionality in
 121047 the Single UNIX Specification Version 4. Its definition subsumes the use of `_POSIX_C_SOURCE`.

121048 An extract of code from a conforming application, that appears before any **#include** statements,
 121049 is given below:

```
121050 #define _XOPEN_SOURCE 700 /* Single UNIX Specification, Version 4 */
121051 #include ...
```

121052 Note that the definition of `_XOPEN_SOURCE` with the value 700 makes the definition of
 121053 `_POSIX_C_SOURCE` redundant and it can safely be omitted.

121054 *B.2.2.2 The Name Space*

121055 The reservation of identifiers is paraphrased from the ISO C standard. The text is included
 121056 because it needs to be part of POSIX.1-2017, regardless of possible changes in future versions of
 121057 the ISO C standard.

121058 These identifiers may be used by implementations, particularly for feature test macros.
 121059 Implementations should not use feature test macro names that might be reasonably used by a
 121060 standard.

121061 Including headers more than once is a reasonably common practice, and it should be carried
 121062 forward from the ISO C standard. More significantly, having definitions in more than one
 121063 header is explicitly permitted. Where the potential declaration is “benign” (the same definition
 121064 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true
 121065 of macros, for example.) In those situations where a repetition is not benign (for example,
 121066 **typedefs**), conditional compilation must be used. The situation actually occurs both within the
 121067 ISO C standard and within POSIX.1: **time_t** should be in `<sys/types.h>`, and the ISO C standard
 121068 mandates that it be in `<time.h>`.

121069 The area of name space pollution *versus* additions to structures is difficult because of the macro
 121070 structure of C. The following discussion summarizes all the various problems with and
 121071 objections to the issue.

121072 Note the phrase “user-defined macro”. Users are not permitted to define macro names (or any
 121073 other name) beginning with “_[A-Z_]””. Thus, the conflict cannot occur for symbols reserved
 121074 to the vendor’s name space, and the permission to add fields automatically applies, without
 121075 qualification, to those symbols.

121076 1. Data structures (and unions) need to be defined in headers by implementations to meet
 121077 certain requirements of POSIX.1 and the ISO C standard.

121078 2. The structures defined by POSIX.1 are typically minimal, and any practical
 121079 implementation would wish to add fields to these structures either to hold additional
 121080 related information or for backwards-compatibility (or both). Future standards (and *de*
 121081 *facto* standards) would also wish to add to these structures. Issues of field alignment
 121082 make it impractical (at least in the general case) to simply omit fields when they are not
 121083 defined by the particular standard involved.

121084 The **dirent** structure is an example of such a minimal structure (although one could argue
 121085 about whether the other fields need visible names). The *st_rdev* field of most
 121086 implementations’ **stat** structure is a common example where extension is needed and
 121087 where a conflict could occur.

121088 3. Fields in structures are in an independent name space, so the addition of such fields
 121089 presents no problem to the C language itself in that such names cannot interact with
 121090 identically named user symbols because access is qualified by the specific structure name.

121091 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols
 121092 added to a structure might be recognized as user-provided macro names at the location
 121093 where the structure is declared. This only can occur if the user-provided name is declared
 121094 as a macro before the header declaring the structure is included. The user’s use of the
 121095 name after the declaration cannot interfere with the structure because the symbol is
 121096 hidden and only accessible through access to the structure. Presumably, the user would
 121097 not declare such a macro if there was an intention to use that field name.

121098 5. Macros from the same or a related header might use the additional fields in the structure,
 121099 and those field names might also collide with user macros. Although this is a less
 121100 frequent occurrence, since macros are expanded at the point of use, no constraint on the
 121101 order of use of names can apply.

121102 6. An “obvious” solution of using names in the reserved name space and then redefining
 121103 them as macros when they should be visible does not work because this has the effect of
 121104 exporting the symbol into the general name space. For example, given a (hypothetical)
 121105 system-provided header `<h.h>`, and two parts of a C program in `a.c` and `b.c`, in header
 121106 `<h.h>`:

```
121107 struct foo {
121108     int __i;
```

```

121109     }
121110     #ifdef _FEATURE_TEST
121111     #define i __i;
121112     #endif

```

121113 **In file a.c:**

```

121114     #include h.h
121115     extern int i;
121116     ...

```

121117 **In file b.c:**

```

121118     extern int i;
121119     ...

```

121120 The symbol that the user thinks of as *i* in both files has an external name of `__i` in **a.c**; the
121121 same symbol *i* in **b.c** has an external name *i* (ignoring any hidden manipulations the
121122 compiler might perform on the names). This would cause a mysterious name resolution
121123 problem when **a.o** and **b.o** are linked.

121124 Simply avoiding definition then causes alignment problems in the structure.

121125 A structure of the form:

```

121126     struct foo {
121127         union {
121128             int __i;
121129             #ifdef _FEATURE_TEST
121130                 int i;
121131             #endif
121132         } __ii;
121133     }

```

121134 does not work because the name of the logical field *i* is `__ii.i`, and introduction of a macro
121135 to restore the logical name immediately reintroduces the problem discussed previously
121136 (although its manifestation might be more immediate because a syntax error would result
121137 if a recursive macro did not cause it to fail first).

121138 7. A more workable solution would be to declare the structure:

```

121139     struct foo {
121140     #ifdef _FEATURE_TEST
121141         int i;
121142     #else
121143         int __i;
121144     #endif
121145     }

```

121146 However, if a macro (particularly one required by a standard) is to be defined that uses
121147 this field, two must be defined: one that uses *i*, the other that uses `__i`. If more than one
121148 additional field is used in a macro and they are conditional on distinct combinations of
121149 features, the complexity goes up as 2^n .

121150 All this leaves a difficult situation: vendors must provide very complex headers to deal with
121151 what is conceptually simple and safe — adding a field to a structure. It is the possibility of user-
121152 provided macros with the same name that makes this difficult.

121153 Several alternatives were proposed that involved constraining the user's access to part of the

121154 name space available to the user (as specified by the ISO C standard). In some cases, this was
 121155 only until all the headers had been included. There were two proposals discussed that failed to
 121156 achieve consensus:

- 121157 1. Limiting it for the whole program.
- 121158 2. Restricting the use of identifiers containing only uppercase letters until after all system
 121159 headers had been included. It was also pointed out that because macros might wish to
 121160 access fields of a structure (and macro expansion occurs totally at point of use) restricting
 121161 names in this way would not protect the macro expansion, and thus the solution was
 121162 inadequate.

121163 It was finally decided that reservation of symbols would occur, but as constrained.

121164 The current wording also allows the addition of fields to a structure, but requires that user
 121165 macros of the same name not interfere. This allows vendors to do one of the following:

121166 Not create the situation (do not extend the structures with user-accessible names or use the
 121167 solution in (7) above)

121168 Extend their compilers to allow some way of adding names to structures and macros safely

121169 There are at least two ways that the compiler might be extended: add new preprocessor
 121170 directives that turn off and on macro expansion for certain symbols (without changing the value
 121171 of the macro) and a function or lexical operation that suppresses expansion of a word. The latter
 121172 seems more flexible, particularly because it addresses the problem in macros as well as in
 121173 declarations.

121174 The following seems to be a possible implementation extension to the C language that will do
 121175 this: any token that during macro expansion is found to be preceded by three '#' symbols shall
 121176 not be further expanded in exactly the same way as described for macros that expand to their
 121177 own name as in Section 3.8.3.4 of the ISO C standard. A vendor may also wish to implement this
 121178 as an operation that is lexically a function, which might be implemented as:

```
121179 #define __safe_name(x) ###x
```

121180 Using a function notation would insulate vendors from changes in standards until such a
 121181 functionality is standardized (if ever). Standardization of such a function would be valuable
 121182 because it would then permit third parties to take advantage of it portably in software they may
 121183 supply.

121184 The symbols that are "explicitly permitted, but not required by POSIX.1-2017" include those
 121185 classified below. (That is, the symbols classified below might, but are not required to, be present
 121186 when `_POSIX_C_SOURCE` is defined to have the value 200809L.)

121187 Symbols in `<limits.h>` and `<unistd.h>` that are defined to indicate support for options or
 121188 limits that are constant at compile-time

121189 Symbols in the name space reserved for the implementation by the ISO C standard

121190 Symbols in a name space reserved for a particular type of extension (for example, type
 121191 names ending with `_t` in `<sys/types.h>`)

121192 Additional members of structures or unions whose names do not reduce the name space
 121193 reserved for applications

121194 Since both implementations and future versions of this standard and other POSIX standards
 121195 may use symbols in the reserved spaces described in these tables, there is a potential for name
 121196 space clashes. To avoid future name space clashes when adding symbols, implementations
 121197 should not use the `posix_`, `POSIX_`, or `_POSIX_` prefixes.

- 121198 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/2 is applied, deleting the entries `POSIX_`,
 121199 `_POSIX_`, and `posix_` from the column of allowed name space prefixes for use by an
 121200 implementation in the first table. The presence of these prefixes was contradicting later text
 121201 which states that: “The prefixes `posix_`, `POSIX_`, and `_POSIX` are reserved for use by XCU
 121202 [Chapter 2](#) (on page 2345) and other POSIX standards. Implementations may add symbols to the
 121203 headers shown in the following table, provided the identifiers ... do not use the reserved
 121204 prefixes `posix_`, `POSIX_`, or `_POSIX`.”
- 121205 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/3 is applied, correcting the reserved
 121206 macro prefix from: “`PRI[a-z]`, `SCN[a-z]`” to: “`PRI[Xa-z]`, `SCN[Xa-z]`” in the second table. The
 121207 change was needed since the ISO C standard allows implementations to define macros of the
 121208 form `PRI` or `SCN` followed by any lowercase letter or 'x' in `<inttypes.h>`. (The
 121209 ISO/IEC 9899:1999 standard, Subclause 7.26.4.)
- 121210 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/4 is applied, adding a new section listing
 121211 reserved names for the `<stdint.h>` header. This change is for alignment with the ISO C standard.
- 121212 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/2 is applied, making it clear that
 121213 implementations are permitted to have symbols with the prefix `_POSIX_` visible in any header.
- 121214 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/3 is applied, updating the table of
 121215 allowed macro prefixes to include the prefix `FP_[A-Z]` for `<math.h>`. This text is added for
 121216 consistency with the `<math.h>` reference page in the Base Definitions volume of POSIX.1-2017
 121217 which permits additional implementation-defined floating-point classifications.
- 121218 Austin Group Interpretation 1003.1-2001 #048 is applied, reserving `SEEK_` in the name space.
- 121219 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0001 [801], XSH/TC2-2008/0002 [780],
 121220 XSH/TC2-2008/0003 [790], XSH/TC2-2008/0004 [780], XSH/TC2-2008/0005 [790],
 121221 XSH/TC2-2008/0006 [782], XSH/TC2-2008/0007 [790], and XSH/TC2-2008/0008 [790] are
 121222 applied.

121223 B.2.3 Error Numbers

- 121224 It was the consensus of the standard developers that to allow the conformance document to state
 121225 that an error occurs and under what conditions, but to disallow a statement that it never occurs,
 121226 does not make sense. It could be implied by the current wording that this is allowed, but to
 121227 reduce the possibility of future interpretation requests, it is better to make an explicit statement.
- 121228 The ISO C standard requires that `errno` be an assignable lvalue. Originally, the definition in
 121229 POSIX.1 was stricter than that in the ISO C standard, `extern int errno`, in order to support
 121230 historical usage. In a multi-threaded environment, implementing `errno` as a global variable
 121231 results in non-deterministic results when accessed. It is required, however, that `errno` work as a
 121232 per-thread error reporting mechanism. In order to do this, a separate `errno` value has to be
 121233 maintained for each thread. The following section discusses the various alternative solutions
 121234 that were considered.
- 121235 In order to avoid this problem altogether for new functions, these functions avoid using `errno`
 121236 and, instead, return the error number directly as the function return value; a return value of zero
 121237 indicates that no error was detected.
- 121238 For any function that can return errors, the function return value is not used for any purpose
 121239 other than for reporting errors. Even when the output of the function is scalar, it is passed
 121240 through a function argument. While it might have been possible to allow some scalar outputs to
 121241 be coded as negative function return values and mixed in with positive error status returns, this
 121242 was rejected—using the return value for a mixed purpose was judged to be of limited use and
 121243 error prone.

121244 Checking the value of *errno* alone is not sufficient to determine the existence or type of an error,
 121245 since it is not required that a successful function call clear *errno*. The variable *errno* should only
 121246 be examined when the return value of a function indicates that the value of *errno* is meaningful.
 121247 In that case, the function is required to set the variable to something other than zero.

121248 The variable *errno* is never set to zero by any function call; to do so would contradict the ISO C
 121249 standard.

121250 POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set
 121251 in certain conditions because many existing applications depend on them. Some error numbers,
 121252 such as [EFAULT], are entirely implementation-defined and are noted as such in their
 121253 description in the ERRORS section. This section otherwise allows wide latitude to the
 121254 implementation in handling error reporting.

121255 Some of the ERRORS sections in POSIX.1-2017 have two subsections. The first:

121256 ``The function shall fail if:``

121257 could be called the ``mandatory`` section.

121258 The second:

121259 ``The function may fail if:``

121260 could be informally known as the ``optional`` section.

121261 Attempting to infer the quality of an implementation based on whether it detects optional error
 121262 conditions is not useful.

121263 Following each one-word symbolic name for an error, there is a description of the error. The
 121264 rationale for some of the symbolic names follows:

121265 [ECANCELED] This spelling was chosen as being more common.

121266 [EFAULT] Most historical implementations do not catch an error and set *errno* when an
 121267 invalid address is given to the functions *wait()*, *time()*, or *times()*. Some
 121268 implementations cannot reliably detect an invalid address. And most systems
 121269 that detect invalid addresses will do so only for a system call, not for a library
 121270 routine.

121271 [EFTYPE] This error code was proposed in earlier proposals as ``Inappropriate operation
 121272 for file type``, meaning that the operation requested is not appropriate for the
 121273 file specified in the function call. This code was proposed, although the same
 121274 idea was covered by [ENOTTY], because the connotations of the name would
 121275 be misleading. It was pointed out that the *fcntl()* function uses the error code
 121276 [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed
 121277 to this code.

121278 [EINTR] POSIX.1 prohibits conforming implementations from restarting interrupted
 121279 system calls of conforming applications unless the SA_RESTART flag is in
 121280 effect for the signal. However, it does not require that [EINTR] be returned
 121281 when another legitimate value may be substituted; for example, a partial
 121282 transfer count when *read()* or *write()* are interrupted. This is only given when
 121283 the signal-catching function returns normally as opposed to returns by
 121284 mechanisms like *longjmp()* or *siglongjmp()*.

121285 [ELOOP] In specifying conditions under which implementations would generate this
 121286 error, the following goals were considered:

121287		To ensure that actual loops are detected, including loops that result from symbolic links across distributed file systems.
121288		
121289		To ensure that during pathname resolution an application can rely on the ability to follow at least {SYMLOOP_MAX} symbolic links in the absence of a loop.
121290		
121291		
121292		To allow implementations to provide the capability of traversing more than {SYMLOOP_MAX} symbolic links in the absence of a loop.
121293		
121294		To allow implementations to detect loops and generate the error prior to encountering {SYMLOOP_MAX} symbolic links.
121295		
121296	[ENAMETOOLONG]	
121297		When a symbolic link is encountered during pathname resolution, the contents of that symbolic link are used to create a new pathname. The standard developers intended to allow, but not require, that implementations enforce the restriction of {PATH_MAX} on the result of this pathname substitution.
121298		
121299		
121300		
121301		
121302		Implementations are allowed, but not required, to treat a pathname longer than {PATH_MAX} passed into the system as an error. Implementations are required to return a pathname (even if it is longer than {PATH_MAX}) when the user supplies a buffer with an interface that specifies the buffer size, as long as the user-supplied buffer is large enough to hold the entire pathname (see XSH <i>getcwd()</i> for an example of this type of interface). Implementations are required to treat a request to pass a pathname longer than {PATH_MAX} from the system to a user-supplied buffer of an unspecified size (usually assumed to be of size {PATH_MAX}) as an error (see XSH <i>realpath()</i> for an example of this type of interface).
121303		
121304		
121305		
121306		
121307		
121308		
121309		
121310		
121311		
121312	[ENOMEM]	The term “main memory” is not used in POSIX.1 because it is implementation-defined.
121313		
121314	[ENOTSUP]	This error code is to be used when an implementation chooses to implement the required functionality of POSIX.1-2017 but does not support optional facilities defined by POSIX.1-2017. In some earlier versions of this standard, the difference between [ENOTSUP] and [ENOSYS] was that [ENOSYS] indicated that the function was not supported at all. This is no longer the case as [ENOSYS] can also be used to indicate non-support of optional functionality for a function that has some required functionality. (See XSH <i>encrypt()</i> .)
121315		
121316		
121317		
121318		
121319		
121320		
121321		
121322	[ENOTTY]	The symbolic name for this error is derived from a time when device control was done by <i>ioctl()</i> and that operation was only permitted on a terminal interface. The term “TTY” is derived from “teletypewriter”, the devices to which this error originally applied.
121323		
121324		
121325		
121326	[EOVERFLOW]	Most of the uses of this error code are related to large file support. Typically, these cases occur on systems which support multiple programming environments with different sizes for off_t , but they may also occur in connection with remote file systems.
121327		
121328		
121329		
121330		In addition, when different programming environments have different widths for types such as int and uid_t , several functions may encounter a condition where a value in a particular environment is too wide to be represented. In that case, this error should be raised. For example, suppose the currently running process has 64-bit int , and file descriptor 9 223 372 036 854 775 807 is
121331		
121332		
121333		
121334		

121335 open and does not have the close-on-exec flag set. If the process then uses
 121336 *execl()* to *exec* a file compiled in a programming environment with 32-bit **int**,
 121337 the call to *execl()* can fail with *errno* set to [E_OVERFLOW]. A similar failure
 121338 can occur with *execl()* if any of the user IDs or any of the group IDs to be
 121339 assigned to the new process image are out of range for the executed file's
 121340 programming environment.

121341 Note, however, that this condition cannot occur for functions that are
 121342 explicitly described as always being successful, such as *getpid()*.

121343 [EPIPE] This condition normally generates the signal SIGPIPE; the error is returned if
 121344 the generation of the signal is suppressed or the signal does not terminate the
 121345 process.

121346 [EROFS] In historical implementations, attempting to *unlink()* or *rmdir()* a mount point
 121347 would generate an [EBUSY] error. An implementation could be envisioned
 121348 where such an operation could be performed without error. In this case, if
 121349 *either* the directory entry or the actual data structures reside on a read-only file
 121350 system, [EROFS] is the appropriate error to generate. (For example, changing
 121351 the link count of a file on a read-only file system could not be done, as is
 121352 required by *unlink()*, and thus an error should be reported.)

121353 Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily
 121354 for consistency with the ISO C standard.

121355 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0009 [496] and XSH/TC2-2008/0010
 121356 [681] are applied.

121357 **Alternative Solutions for Per-Thread *errno***

121358 The usual implementation of *errno* as a single global variable does not work in a multi-threaded
 121359 environment. In such an environment, a thread may make a POSIX.1 call and get a -1 error
 121360 return, but before that thread can check the value of *errno*, another thread might have made a
 121361 second POSIX.1 call that also set *errno*. This behavior is unacceptable in robust programs. There
 121362 were a number of alternatives that were considered for handling the *errno* problem:

121363 Implement *errno* as a per-thread integer variable.

121364 Implement *errno* as a service that can access the per-thread error number.

121365 Change all POSIX.1 calls to accept an extra status argument and avoid setting *errno*.

121366 Change all POSIX.1 calls to raise a language exception.

121367 The first option offers the highest level of compatibility with existing practice but requires
 121368 special support in the linker, compiler, and/or virtual memory system to support the new
 121369 concept of thread private variables. When compared with current practice, the third and fourth
 121370 options are much cleaner, more efficient, and encourage a more robust programming style, but
 121371 they require new versions of all of the POSIX.1 functions that might detect an error. The second
 121372 option offers compatibility with existing code that uses the `<errno.h>` header to define the
 121373 symbol *errno*. In this option, *errno* may be a macro defined:

```
121374 #define errno (*__errno())
121375 extern int *__errno();
```

121376 This option may be implemented as a per-thread variable whereby an *errno* field is allocated in
 121377 the user space object representing a thread, and whereby the function *__errno()* makes a system
 121378 call to determine the location of its user space object and returns the address of the *errno* field of
 121379 that object. Another implementation, one that avoids calling the kernel, involves allocating

121380 stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a
 121381 pointer to the thread object that uses that chunk. The `__errno()` function then looks at the stack
 121382 pointer, determines the chunk number, and uses that as an index into the chunk table to find its
 121383 thread object and thus its private value of `errno`. On most architectures, this can be done in four
 121384 to five instructions. Some compilers may wish to implement `__errno()` inline to improve
 121385 performance.

121386 **Disallowing Return of the [EINTR] Error Code**

121387 Many blocking interfaces defined by POSIX.1-2017 may return [EINTR] if interrupted during
 121388 their execution by a signal handler. Blocking interfaces introduced under the threads
 121389 functionality do not have this property. Instead, they require that the interface appear to be
 121390 atomic with respect to interruption. In particular, clients of blocking interfaces need not handle
 121391 any possible [EINTR] return as a special case since it will never occur. If it is necessary to restart
 121392 operations or complete incomplete operations following the execution of a signal handler, this is
 121393 handled by the implementation, rather than by the application.

121394 Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a
 121395 frequent source of often unreproducible bugs, and it adds no compelling value to the available
 121396 functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not
 121397 use this paradigm. In particular, in none of the functions `flockfile()`, `pthread_cond_timedwait()`,
 121398 `pthread_cond_wait()`, `pthread_join()`, `pthread_mutex_lock()`, and `sigwait()` did providing [EINTR]
 121399 returns add value, or even particularly make sense. Thus, these functions do not provide for an
 121400 [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied
 121401 to `sem_wait()`, `sem_trywait()`, `sigwaitinfo()`, and `sigtimedwait()`, but implementations are
 121402 permitted to return [EINTR] error codes for these functions for compatibility with earlier
 121403 versions of this standard. Applications cannot rely on calls to these functions returning [EINTR]
 121404 error codes when signals are delivered to the calling thread, but they should allow for the
 121405 possibility.

121406 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and
 121407 [EOPNOTSUPP] to be the same values.

121408 *B.2.3.1 Additional Error Numbers*

121409 The ISO C standard defines the name space for implementations to add additional error
 121410 numbers.

121411 **B.2.4 Signal Concepts**

121412 Historical implementations of signals, using the `signal()` function, have shortcomings that make
 121413 them unreliable for many application uses. Because of this, a new signal mechanism, based very
 121414 closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

121415 **Signal Names**

121416 The restriction on the actual type used for `sigset_t` is intended to guarantee that these objects can
 121417 always be assigned, have their address taken, and be passed as parameters by value. It is not
 121418 intended that this type be a structure including pointers to other data structures, as that could
 121419 impact the portability of applications performing such operations. A reasonable implementation
 121420 could be a structure containing an array of some integer type.

121421 The signals described in POSIX.1-2017 must have unique values so that they may be named as
 121422 parameters of `case` statements in the body of a C-language `switch` clause. However,

121423 implementation-defined signals may have values that overlap with each other or with signals
121424 specified in POSIX.1-2017. An example of this is SIGABRT, which traditionally overlaps some
121425 other signal, such as SIGIOT.

121426 SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit
121427 use of the *kill()* function, although some implementations generate SIGKILL under
121428 extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill*
121429 command.

121430 The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1
121431 because their behavior is implementation-defined and could not be adequately categorized.
121432 Conforming implementations may deliver these signals, but must document the circumstances
121433 under which they are delivered and note any restrictions concerning their delivery. The signals
121434 SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from
121435 programming errors. They were included in POSIX.1 because they do indicate three relatively
121436 well-categorized conditions. They are all defined by the ISO C standard and thus would have to
121437 be defined by any system with an ISO C standard binding, even if not explicitly included in
121438 POSIX.1.

121439 There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or
121440 masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or
121441 SIGFPE. They will generally be generated by the system only in cases of programming errors.
121442 While it may be desirable for some robust code (for example, a library routine) to be able to
121443 detect and recover from programming errors in other code, these signals are not nearly sufficient
121444 for that purpose. One portable use that does exist for these signals is that a command interpreter
121445 can recognize them as the cause of termination of a process (with *wait()*) and print an
121446 appropriate message. The mnemonic tags for these signals are derived from their PDP-11 origin.

121447 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control
121448 and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control
121449 shells to detect children that have terminated or, as in 4.2 BSD, stopped.

121450 Some implementations, including System V, have a signal named SIGCLD, which is similar to
121451 SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both
121452 names. POSIX.1 carefully specifies ways in which conforming applications can avoid the
121453 semantic differences between the two different implementations. The name SIGCHLD was
121454 chosen for POSIX.1 because most current application usages of it can remain unchanged in
121455 conforming applications. SIGCLD in System V has more cases of semantics that POSIX.1 does
121456 not specify, and thus applications using it are more likely to require changes in addition to the
121457 name change.

121458 The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of
121459 exceptional behavior and are described as “reserved as application-defined” so that such use is
121460 not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when
121461 explicitly requested by *kill()*. It is recommended that libraries not use these two signals, as such
121462 use in libraries could interfere with their use by applications calling the libraries. If such use is
121463 unavoidable, it should be documented. It is prudent for non-portable libraries to use non-
121464 standard signals to avoid conflicts with use of standard signals by portable libraries.

121465 There is no portable way for an application to catch or ignore non-standard signals. Some
121466 implementations define the range of signal numbers, so applications can install signal-catching
121467 functions for all of them. Unfortunately, implementation-defined signals often cause problems
121468 when caught or ignored by applications that do not understand the reason for the signal. While
121469 the desire exists for an application to be more robust by handling all possible signals (even those
121470 only generated by *kill()*), no existing mechanism was found to be sufficiently portable to include
121471 in POSIX.1. The value of such a mechanism, if included, would be diminished given that

121472 SIGKILL would still not be catchable.

121473 A number of new signal numbers are reserved for applications because the two user signals
121474 defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is
121475 specified, rather than an enumeration of additional reserved signal names, because different
121476 applications and application profiles will require a different number of application signals. It is
121477 not desirable to burden all application domains and therefore all implementations with the
121478 maximum number of signals required by all possible applications. Note that in this context,
121479 signal numbers are essentially different signal priorities.

121480 The relatively small number of required additional signals, `{_POSIX_RTSIG_MAX}`, was chosen
121481 so as not to require an unreasonably large signal mask/set. While this number of signals
121482 defined in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing
121483 implementations define many more signals than are specified in POSIX.1 and, in fact, many
121484 implementations have already exceeded 32 signals (including the “null signal”). Support of
121485 `{_POSIX_RTSIG_MAX}` additional signals may push some implementation over the single 32-bit
121486 word line, but is unlikely to push any implementations that are already over that line beyond
121487 the 64-signal line.

121488 B.2.4.1 Signal Generation and Delivery

121489 The terms defined in this section are not used consistently in documentation of historical
121490 systems. Each signal can be considered to have a lifetime beginning with generation and ending
121491 with delivery or acceptance. The POSIX.1 definition of “delivery” does not exclude ignored
121492 signals; this is considered a more consistent definition. This revised text in several parts of
121493 POSIX.1-2017 clarifies the distinct semantics of asynchronous signal delivery and synchronous
121494 signal acceptance. The previous wording attempted to categorize both under the term
121495 “delivery”, which led to conflicts over whether the effects of asynchronous signal delivery
121496 applied to synchronous signal acceptance.

121497 Signals generated for a process are delivered to only one thread. Thus, if more than one thread
121498 is eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the
121499 implementation both to allow the widest possible range of conforming implementations and to
121500 give implementations the freedom to deliver the signal to the “easiest possible” thread should
121501 there be differences in ease of delivery between different threads.

121502 Note that should multiple delivery among cooperating threads be required by an application,
121503 this can be trivially constructed out of the provided single-delivery semantics. The construction
121504 of a `sigwait_multiple()` function that accomplishes this goal is presented with the rationale for
121505 `sigwaitinfo()`.

121506 Implementations should deliver unblocked signals as soon after they are generated as possible.
121507 However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in
121508 `kill()` and `sigprocmask()`. Even on systems with prompt delivery, scheduling of higher priority
121509 processes is always likely to cause delays.

121510 In general, the interval between the generation and delivery of unblocked signals cannot be
121511 detected by an application. Thus, references to pending signals generally apply to blocked,
121512 pending signals. An implementation registers a signal as pending on the process when no
121513 thread has the signal unblocked and there are no threads blocked in a `sigwait()` function for that
121514 signal. Thereafter, the implementation delivers the signal to the first thread that unblocks the
121515 signal or calls a `sigwait()` function on a signal set containing this signal rather than choosing the
121516 recipient thread at the time the signal is sent.

121517 In the 4.3 BSD system, signals that are blocked and set to `SIG_IGN` are discarded immediately
121518 upon generation. For a signal that is ignored as its default action, if the action is `SIG_DFL` and

121519 the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in
 121520 System V Release 3 (two other implementations that support a somewhat similar signal
 121521 mechanism), all ignored blocked signals remain pending if generated. Because it is not normally
 121522 useful for an application to simultaneously ignore and block the same signal, it was unnecessary
 121523 for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

121524 There is one case in some historical implementations where an unblocked, pending signal does
 121525 not remain pending until it is delivered. In the System V implementation of *signal()*, pending
 121526 signals are discarded when the action is set to SIG_DFL or a signal-catching routine (as well as
 121527 to SIG_IGN). Except in the case of setting SIGCHLD to SIG_DFL, implementations that do this
 121528 do not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this,
 121529 but these statements were redundant due to the requirement that functions defined by POSIX.1
 121530 not change attributes of processes defined by POSIX.1 except as explicitly stated.

121531 POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are
 121532 delivered is unspecified. This order has not been explicitly specified in historical
 121533 implementations, but has remained quite consistent and been known to those familiar with the
 121534 implementations. Thus, there have been cases where applications (usually system utilities) have
 121535 been written with explicit or implicit dependencies on this order. Implementors and others
 121536 porting existing applications may need to be aware of such dependencies.

121537 When there are multiple pending signals that are not blocked, implementations should arrange
 121538 for the delivery of all signals at once, if possible. Some implementations stack calls to all pending
 121539 signal-catching routines, making it appear that each signal-catcher was interrupted by the next
 121540 signal. In this case, the implementation should ensure that this stacking of signals does not
 121541 violate the semantics of the signal masks established by *sigaction()*. Other implementations
 121542 process at most one signal when the operating system is entered, with remaining signals saved
 121543 for later delivery. Although this practice is widespread, this behavior is neither standardized
 121544 nor endorsed. In either case, implementations should attempt to deliver signals associated with
 121545 the current state of the process (for example, SIGFPE) before other signals, if possible.

121546 In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if
 121547 blocking or ignoring this signal prevented it from continuing a stopped process, such a process
 121548 could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block
 121549 SIGCONT during execution of its signal-catching function when it is caught, creating exactly
 121550 this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring
 121551 and blocking it, but this limitation led to objections. The consensus was to require that
 121552 SIGCONT always continue a stopped process when generated. This removed the need to
 121553 disallow ignoring or explicit blocking of the signal; note that SIG_IGN and SIG_DFL are
 121554 equivalent for SIGCONT.

121555 B.2.4.2 Realtime Signal Generation and Delivery

121556 The realtime signals functionality is required in this version of the standard for the following
 121557 reasons:

121558 The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous
 121559 event notifications to specify the notification mechanism to use and other information
 121560 needed by the notification mechanism. POSIX.1-2017 defines only three symbolic values
 121561 for the notification mechanism:

121562 †SIGEV_NONE is used to indicate that no notification is required when the event
 121563 occurs. This is useful for applications that use asynchronous I/O with polling for
 121564 completion.

- 121565 ‡ **SIGEV_SIGNAL** indicates that a signal is generated when the event occurs.
- 121566 ‡ **SIGEV_THREAD** provides for “callback functions” for asynchronous notifications
121567 done by a function call within the context of a new thread. This provides a multi-
121568 threaded process with a more natural means of notification than signals.
- 121569 The primary difficulty with previous notification approaches has been to specify the
121570 environment of the notification routine.
- 121571 ‡ **One** approach is to limit the notification routine to call only functions permitted in a
121572 signal handler. While the list of permissible functions is clearly stated, this is overly
121573 restrictive.
- 121574 ‡ **A second** approach is to define a new list of functions or classes of functions that are
121575 explicitly permitted or not permitted. This would give a programmer more lists to
121576 deal with, which would be awkward.
- 121577 ‡ **The third** approach is to define completely the environment for execution of the
121578 notification function. A clear definition of an execution environment for notification
121579 is provided by executing the notification function in the environment of a newly
121580 created thread.
- 121581 Implementations may support additional notification mechanisms by defining new values
121582 for *sigev_notify*.
- 121583 For a notification type of **SIGEV_SIGNAL**, the other members of the **sigevent** structure
121584 defined by POSIX.1-2017 specify the realtime signal—that is, the signal number and
121585 application-defined value that differentiates between occurrences of signals with the same
121586 number ‡ that will be generated when the event occurs. The structure is defined in
121587 **<signal.h>**, even though the structure is not directly used by any of the signal functions,
121588 because it is part of the signals interface used by the POSIX.1b “client functions”. When the
121589 client functions include **<signal.h>** to define the signal names, the **sigevent** structure will
121590 also be defined.
- 121591 An application-defined value passed to the signal handler is used to differentiate between
121592 different “events” instead of requiring that the application use different signal numbers for
121593 several reasons:
- 121594 ‡ **Realtime** applications potentially handle a very large number of different events.
121595 Requiring that implementations support a correspondingly large number of distinct
121596 signal numbers will adversely impact the performance of signal delivery because the
121597 signal masks to be manipulated on entry and exit to the handlers will become large.
- 121598 ‡ **Event** notifications are prioritized by signal number (the rationale for this is
121599 explained in the following paragraphs) and the use of different signal numbers to
121600 differentiate between the different event notifications overloads the signal number
121601 more than has already been done. It also requires that the application developer
121602 make arbitrary assignments of priority to events that are logically of equal priority.
- 121603 A union is defined for the application-defined value so that either an integer constant or a
121604 pointer can be portably passed to the signal-catching function. On some architectures a
121605 pointer cannot be cast to an **int** and *vice versa*.
- 121606 Use of a structure here with an explicit notification type discriminant rather than explicit
121607 parameters to realtime functions, or embedded in other realtime structures, provides for
121608 future extensions to POSIX.1-2017. Additional, perhaps more efficient, notification
121609 mechanisms can be supported for existing realtime function interfaces, such as timers and
121610 asynchronous I/O, by extending the **sigevent** structure appropriately. The existing
121611 realtime function interfaces will not have to be modified to use any such new notification

121612 mechanism. The revised text concerning the SIGEV_SIGNAL value makes consistent the
 121613 semantics of the members of the **sigevent** structure, particularly in the definitions of
 121614 *lio_listio()* and *aio_fsync()*. For uniformity, other revisions cause this specification to be
 121615 referred to rather than inaccurately duplicated in the descriptions of functions and
 121616 structures using the **sigevent** structure. The revised wording does not relax the
 121617 requirement that the signal number be in the range SIGRTMIN to SIGRTMAX to guarantee
 121618 queuing and passing of the application value, since that requirement is still implied by the
 121619 signal names.

121620 POSIX.1-2017 is intentionally vague on whether “non-realtime” signal-generating
 121621 mechanisms can result in a **siginfo_t** being supplied to the handler on delivery. In one
 121622 existing implementation, a **siginfo_t** is posted on signal generation, even though the
 121623 implementation does not support queuing of multiple occurrences of a signal. It is not the
 121624 intent of POSIX.1-2017 to preclude this, independent of the mandate to define signals that
 121625 do support queuing. Any interpretation that appears to preclude this is a mistake in the
 121626 reading or writing of the standard.

121627 Signals handled by realtime signal handlers might be generated by functions or conditions
 121628 that do not allow the specification of an application-defined value and do not queue.
 121629 POSIX.1-2017 specifies the *si_code* member of the **siginfo_t** structure used in existing
 121630 practice and defines additional codes so that applications can detect whether an
 121631 application-defined value is present or not. The code SI_USER for *kill()*-generated signals
 121632 is adopted from existing practice.

121633 The *sigaction()* *sa_flags* value SA_SIGINFO tells the implementation that the signal-
 121634 catching function expects two additional arguments. When the flag is not set, a single
 121635 argument, the signal number, is passed as specified by POSIX.1-2017. Although
 121636 POSIX.1-2017 does not explicitly allow the *info* argument to the handler function to be
 121637 NULL, this is existing practice. This provides for compatibility with programs whose
 121638 signal-catching functions are not prepared to accept the additional arguments.
 121639 POSIX.1-2017 is explicitly unspecified as to whether signals actually queue when
 121640 SA_SIGINFO is not set for a signal, as there appear to be no benefits to applications in
 121641 specifying one behavior or another. One existing implementation queues a **siginfo_t** on
 121642 each signal generation, unless the signal is already pending, in which case the
 121643 implementation discards the new **siginfo_t**; that is, the queue length is never greater than
 121644 one. This implementation only examines SA_SIGINFO on signal delivery, discarding the
 121645 queued **siginfo_t** if its delivery was not requested.

121646 The third argument to the signal-catching function, *context*, is left undefined by
 121647 POSIX.1-2017, but is specified in the interface because it matches existing practice for the
 121648 SA_SIGINFO flag. It was considered undesirable to require a separate implementation for
 121649 SA_SIGINFO for POSIX conformance on implementations that already support the two
 121650 additional parameters.

121651 The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX
 121652 first, when multiple unblocked signals are pending, results from several considerations:

121653 ‡ A method is required to prioritize event notifications. The signal number was chosen
 121654 instead of, for instance, associating a separate priority with each request, because an
 121655 implementation has to check pending signals at various points and select one for
 121656 delivery when more than one is pending. Specifying a selection order is the minimal
 121657 additional semantic that will achieve prioritized delivery. If a separate priority were
 121658 to be associated with queued signals, it would be necessary for an implementation to
 121659 search all non-empty, non-blocked signal queues and select from among them the
 121660 pending signal with the highest priority. This would significantly increase the cost of
 121661 and decrease the determinism of signal delivery.

121662 † Given the specified selection of the lowest numeric unblocked pending signal,
 121663 preemptive priority signal delivery can be achieved using signal numbers and signal
 121664 masks by ensuring that the *sa_mask* for each signal number blocks all signals with a
 121665 higher numeric value.

121666 For realtime applications that want to use only the newly defined realtime signal
 121667 numbers without interference from the standard signals, this can be achieved by
 121668 blocking all of the standard signals in the thread signal mask and in the *sa_mask*
 121669 installed by the signal action for the realtime signal handlers.

121670 POSIX.1-2017 explicitly leaves unspecified the ordering of signals outside of the range of
 121671 realtime signals and the ordering of signals within this range with respect to those outside
 121672 the range. It was believed that this would unduly constrain implementations or standards
 121673 in the future definition of new signals.

121674 B.2.4.3 Signal Actions

121675 Early proposals mentioned SIGCONT as a second exception to the rule that signals are not
 121676 delivered to stopped processes until continued. Because POSIX.1-2017 now specifies that
 121677 SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is
 121678 not prevented because a process is stopped, even without an explicit exception to this rule.

121679 Ignoring a signal by setting the action to SIG_IGN (or SIG_DFL for signals whose default action
 121680 is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking
 121681 such a function will interrupt certain system functions that block processes (for example, *wait()*,
 121682 *sigsuspend()*, *pause()*, *read()*, *write()*) while ignoring a signal has no such effect on the process.

121683 Historical implementations discard pending signals when the action is set to SIG_IGN.
 121684 However, they do not always do the same when the action is set to SIG_DFL and the default
 121685 action is to ignore the signal. POSIX.1-2017 requires this for the sake of consistency and also for
 121686 completeness, since the only signal this applies to is SIGCHLD, and POSIX.1-2017 disallows
 121687 setting its action to SIG_IGN.

121688 Some implementations (System V, for example) assign different semantics for SIGCLD
 121689 depending on whether the action is set to SIG_IGN or SIG_DFL. Since POSIX.1 requires that the
 121690 default action for SIGCHLD be to ignore the signal, applications should always set the action to
 121691 SIG_DFL in order to avoid SIGCHLD.

121692 Whether or not an implementation allows SIG_IGN as a SIGCHLD disposition to be inherited
 121693 across a call to one of the *exec* family of functions or *posix_spawn()* is explicitly left as
 121694 unspecified. This change was made as a result of IEEE PASC Interpretation 1003.1 #132, and
 121695 permits the implementation to decide between the following alternatives:

121696 Unconditionally leave SIGCHLD set to SIG_IGN, in which case the implementation would
 121697 not allow applications that assume inheritance of SIG_DFL to conform to POSIX.1-2017
 121698 without change. The implementation would, however, retain an ability to control
 121699 applications that create child processes but never call on the *wait* family of functions,
 121700 potentially filling up the process table.

121701 Unconditionally reset SIGCHLD to SIG_DFL, in which case the implementation would
 121702 allow applications that assume inheritance of SIG_DFL to conform. The implementation
 121703 would, however, lose an ability to control applications that spawn child processes but
 121704 never reap them.

121705 Provide some mechanism, not specified in POSIX.1-2017, to control inherited SIGCHLD
 121706 dispositions.

121707 Some implementations (System V, for example) will deliver a SIGCLD signal immediately when

121708 a process establishes a signal-catching function for SIGCHLD when that process has a child that
 121709 has already terminated. Other implementations, such as 4.3 BSD, do not generate a new
 121710 SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action
 121711 for the SIGCHLD signal while it has any outstanding children. However, it is not always
 121712 possible for a process to avoid this; for example, shells sometimes start up processes in pipelines
 121713 with other processes from the pipeline as children. Processes that cannot ensure that they have
 121714 no children when altering the signal action for SIGCHLD thus need to be prepared for, but not
 121715 depend on, generation of an immediate SIGCHLD signal.

121716 The default action of the stop signals (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is to stop a
 121717 process that is executing. If a stop signal is delivered to a process that is already stopped, it has
 121718 no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the
 121719 signal, the signal will never be delivered to the process since the process must receive a
 121720 SIGCONT, which discards all pending stop signals, in order to continue executing.

121721 The SIGCONT signal continues a stopped process even if SIGCONT is blocked (or ignored).
 121722 However, if a signal-catching routine has been established for SIGCONT, it will not be entered
 121723 until SIGCONT is unblocked.

121724 If a process in an orphaned process group stops, it is no longer under the control of a job control
 121725 shell and hence would not normally ever be continued. Because of this, orphaned processes that
 121726 receive terminal-related stop signals (SIGTSTP, SIGTTIN, SIGTTOU, but not SIGSTOP) must not
 121727 be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As
 121728 SIGSTOP is sent only via *kill()*, it is assumed that the process or user sending a SIGSTOP can
 121729 send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an
 121730 extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is
 121731 overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD
 121732 also does this for orphaned processes (processes whose parent has terminated) rather than for
 121733 members of orphaned process groups; this is less desirable because job control shells manage
 121734 process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for
 121735 processes in orphaned process groups as a direct result of activity on a terminal, preventing
 121736 infinite loops when *read()* and *write()* calls generate signals that are discarded; see [Section](#)
 121737 [A.11.1.4](#) (on page 3551). A similar restriction on the generation of SIGTSTP was considered, but
 121738 that would be unnecessary and more difficult to implement due to its asynchronous nature.

121739 Although POSIX.1 requires that signal-catching functions be called with only one argument,
 121740 there is nothing to prevent conforming implementations from extending POSIX.1 to pass
 121741 additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile
 121742 and execute correctly. Most historical implementations do, in fact, pass additional, signal-
 121743 specific arguments to certain signal-catching routines.

121744 There was a proposal to change the declared type of the signal handler to:

```
121745 void func (int sig, ...);
```

121746 The usage of ellipses ("...") is ISO C standard syntax to indicate a variable number of
 121747 arguments. Its use was intended to allow the implementation to pass additional information to
 121748 the signal handler in a standard manner.

121749 Unfortunately, this construct would require all signal handlers to be defined with this syntax
 121750 because the ISO C standard allows implementations to use a different parameter passing
 121751 mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing
 121752 signal handlers in all existing applications would have to be changed to use the variable syntax
 121753 in order to be standard and portable. This is in conflict with the goal of Minimal Changes to
 121754 Existing Application Code.

121755 When terminating a process from a signal-catching function, processes should be aware of any

121756 interpretation that their parent may make of the status returned by *wait()*, *waitid()*, or *waitpid()*.
121757 In particular, a signal-catching function should not call *exit(0)* or *_exit(0)* unless it wants to
121758 indicate successful termination. A non-zero argument to *exit()* or *_exit()* can be used to indicate
121759 unsuccessful termination. Alternatively, the process can use *kill()* to send itself a fatal signal
121760 (first ensuring that the signal is set to the default action and not blocked). See also the
121761 RATIONALE section of the *_exit()* function.

121762 The behavior of *unsafe* functions, as defined by this section, is undefined when they are called
121763 from (or after a *longjmp()* or *siglongjmp()* out of) signal-catching functions in certain
121764 circumstances. The behavior of *async-signal-safe* functions, as defined by this section, is as
121765 specified by POSIX.1, regardless of invocation from a signal-catching function. This is the only
121766 intended meaning of the statement that *async-signal-safe* functions may be used in signal-
121767 catching functions without restriction. Applications must still consider all effects of such
121768 functions on such things as data structures, files, and process state. In particular, application
121769 developers need to consider the restrictions on interactions when interrupting *sleep()* (see
121770 *sleep()*) and interactions among multiple handles for a file description. The fact that any specific
121771 function is listed as *async-signal-safe* does not necessarily mean that invocation of that function
121772 from a signal-catching function is recommended.

121773 In order to prevent errors arising from interrupting non-*async-signal-safe* function calls,
121774 applications should protect calls to these functions either by blocking the appropriate signals or
121775 through the use of some programmatic semaphore. POSIX.1 does not address the more general
121776 problem of synchronizing access to shared data structures. Note in particular that even the
121777 “safe” functions may modify the global variable *errno*; the signal-catching function may want to
121778 save and restore its value. The same principles apply to the *async-signal-safety* of application
121779 routines and asynchronous data access.

121780 Note that although *longjmp()* and *siglongjmp()* are in the list of *async-signal-safe* functions, there
121781 are restrictions on subsequent behavior after the function is called from a signal-catching
121782 function. This is because the code executing after *longjmp()* or *siglongjmp()* can call any unsafe
121783 functions with the same danger as calling those unsafe functions directly from the signal
121784 handler. Applications that use *longjmp()* or *siglongjmp()* out of signal handlers require rigorous
121785 protection in order to be portable. Many of the other functions that are excluded from the list are
121786 traditionally implemented using either the C language *malloc()* or *free()* functions or the ISO C
121787 standard I/O library, both of which traditionally use data structures in a non-*async-signal-safe*
121788 manner. Because any combination of different functions using a common data structure can
121789 cause *async-signal-safety* problems, POSIX.1 does not define the behavior when any unsafe
121790 function is called in (or after a *longjmp()* or *siglongjmp()* out of) a signal handler that interrupts
121791 any unsafe function or the non-*async-signal-safe* processing equivalent to *exit()* that is
121792 performed after return from the initial call to *main()*.

121793 The only realtime extension to signal actions is the addition of the additional parameters to the
121794 signal-catching function. This extension has been explained and motivated in the previous
121795 section. In making this extension, though, developers of POSIX.1b ran into issues relating to
121796 function prototypes. In response to input from the POSIX.1 standard developers, members were
121797 added to the **sigaction** structure to specify function prototypes for the newer signal-catching
121798 function specified by POSIX.1b. These members follow changes that are being made to POSIX.1.
121799 Note that POSIX.1-2017 explicitly states that these fields may overlap so that a union can be
121800 defined. This enabled existing implementations of POSIX.1 to maintain binary-compatibility
121801 when these extensions were added.

121802 The **siginfo_t** structure was adopted for passing the application-defined value to match existing
121803 practice, but the existing practice has no provision for an application-defined value, so this was
121804 added. Note that POSIX normally reserves the “_t” type designation for opaque types. The
121805 **siginfo_t** structure breaks with this convention to follow existing practice and thus promote

121806 portability.

121807 POSIX.1-2017 specifies several values for the *si_code* member of the **siginfo_t** structure. Some
 121808 were introduced in POSIX.1b; others were XSI functionality in the Single UNIX Specification,
 121809 Version 2 and Version 3, that has now become Base functionality. Historically, an *si_code* value of
 121810 less than or equal to zero indicated that the signal was generated by a process via the *kill()*
 121811 function, and values of *si_code* that provided additional information for implementation-
 121812 generated signals, such as SIGFPE or SIGSEGV, were all positive. This functionality is partially
 121813 specified for XSI systems in that if *si_code* is less than or equal to zero, the signal was generated
 121814 by a process. However, since POSIX.1b did not specify that SI_USER (or SI_QUEUE) had a value
 121815 less than or equal to zero, it is not true that when the signal is generated by a process, the value
 121816 of *si_code* will always be less than or equal to zero. XSI applications should check whether *si_code*
 121817 is SI_USER or SI_QUEUE in addition to checking whether it is less than or equal to zero.
 121818 Applications on systems that do not support the XSI option should just check for SI_USER and
 121819 SI_QUEUE.

121820 If an implementation chooses to define additional values for *si_code*, these values have to be
 121821 different from the values of the non-signal-specific symbols specified by POSIX.1-2017. This will
 121822 allow conforming applications to differentiate between signals generated by standard events
 121823 and those generated by other implementation events in a manner compatible with existing
 121824 practice.

121825 The unique values of *si_code* for the POSIX.1b asynchronous events have implications for
 121826 implementations of, for example, asynchronous I/O or message passing in user space library
 121827 code. Such an implementation will be required to provide a hidden interface to the signal
 121828 generation mechanism that allows the library to specify the standard values of *si_code*.

121829 POSIX.1-2017 also specifies additional members of **siginfo_t**, beyond those that were in
 121830 POSIX.1b. Like the *si_code* values mentioned above, these were XSI functionality in the Single
 121831 UNIX Specification, Version 2 and Version 3, that has now become Base functionality. They
 121832 provide additional information when *si_code* has one of the values that moved from XSI to Base.

121833 Although it is not explicitly visible to applications, there are additional semantics for signal
 121834 actions implied by queued signals and their interaction with other POSIX.1b realtime functions.
 121835 Specifically:

121836 It is not necessary to queue signals whose action is SIG_IGN.

121837 For implementations that support POSIX.1b timers, some interaction with the timer
 121838 functions at signal delivery is implied to manage the timer overrun count.

121839 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/5 is applied, reordering the RTS shaded
 121840 text under the third and fourth paragraphs of the SIG_DFL description. This corrects an earlier
 121841 editorial error in this section.

121842 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/6 is applied, adding the *abort()* function
 121843 to the list of async-signal-safe functions.

121844 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/4 is applied, adding the *socketmark()*
 121845 function to the list of functions that shall be either reentrant or non-interruptible by signals and
 121846 shall be async-signal-safe.

121847 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0011 [690], XSH/TC2-2008/0012 [516],
 121848 XSH/TC2-2008/0013 [692], XSH/TC2-2008/0014 [615], XSH/TC2-2008/0015 [516], and
 121849 XSH/TC2-2008/0016 [807] are applied.

121850 B.2.4.4 *Signal Effects on Other Functions*

121851 The most common behavior of an interrupted function after a signal-catching function returns is
 121852 for the interrupted function to give an [EINTR] error unless the SA_RESTART flag is in effect for
 121853 the signal. However, there are a number of specific exceptions, including *sleep()* and certain
 121854 situations with *read()* and *write()*.

121855 The historical implementations of many functions defined by POSIX.1-2017 are not interruptible,
 121856 but delay delivery of signals generated during their execution until after they complete. This is
 121857 never a problem for functions that are guaranteed to complete in a short (imperceptible to a
 121858 human) period of time. It is normally those functions that can suspend a process indefinitely or
 121859 for long periods of time (for example, *wait()*, *pause()*, *sigsuspend()*, *sleep()*, or *read()/write()* on a
 121860 slow device like a terminal) that are interruptible. This permits applications to respond to
 121861 interactive signals or to set timeouts on calls to most such functions with *alarm()*. Therefore,
 121862 implementations should generally make such functions (including ones defined as extensions)
 121863 interruptible.

121864 Functions not mentioned explicitly as interruptible may be so on some implementations,
 121865 possibly as an extension where the function gives an [EINTR] error. There are several functions
 121866 (for example, *getpid()*, *getuid()*) that are specified as never returning an error, which can thus
 121867 never be extended in this way.

121868 If a signal-catching function returns while the SA_RESTART flag is in effect, an interrupted
 121869 function is restarted at the point it was interrupted. Conforming applications cannot make
 121870 assumptions about the internal behavior of interrupted functions, even if the functions are
 121871 async-signal-safe. For example, suppose the *read()* function is interrupted with SA_RESTART in
 121872 effect, the signal-catching function closes the file descriptor being read from and returns, and the
 121873 *read()* function is then restarted; in this case the application cannot assume that the *read()*
 121874 function will give an [EBADF] error, since *read()* might have checked the file descriptor for
 121875 validity before being interrupted.

121876 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0017 [807] is applied.

121877 **B.2.5 Standard I/O Streams**

121878 Although the ISO C standard guarantees that, at program start-up, *stdin* is open for reading and
 121879 *stdout* and *stderr* are open for writing, this guarantee is contingent (as are all guarantees made by
 121880 the ISO C and POSIX standards) on the program being executed in a conforming environment.
 121881 Programs executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not
 121882 open for writing are executed in a non-conforming environment. Application writers are warned
 121883 (in *exec*, *posix_spawn()*, and Section C.2.7, on page 3735) not to execute a standard utility or a
 121884 conforming application with file descriptor 0 not open for reading or with file descriptor 1 or 2
 121885 not open for writing.

121886 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0018 [608] is applied.

121887 B.2.5.1 *Interaction of File Descriptors and Standard I/O Streams*

121888 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0019 [480] is applied.

121889 B.2.5.2 *Stream Orientation and Encoding Rules*

121890 There is no additional rationale provided for this section.

121891 **B.2.6 STREAMS**

121892 STREAMS are included into POSIX.1-2017 as part of the alignment with the Single UNIX
121893 Specification, but marked as an option in recognition that not all systems may wish to
121894 implement the facility. The option within POSIX.1-2017 is denoted by the XSR margin marker.
121895 The standard developers made this option independent of the XSI option. In this version of the
121896 standard this option is marked obsolescent.

121897 STREAMS are a method of implementing network services and other character-based
121898 input/output mechanisms, with the STREAM being a full-duplex connection between a process
121899 and a device. STREAMS provides direct access to protocol modules, and optional protocol
121900 modules can be interposed between the process-end of the STREAM and the device-driver at the
121901 device-end of the STREAM. Pipes can be implemented using the STREAMS mechanism, so they
121902 can provide process-to-process as well as process-to-device communications.

121903 This section introduces STREAMS I/O, the message types used to control them, an overview of
121904 the priority mechanism, and the interfaces used to access them.

121905 B.2.6.1 *Accessing STREAMS*

121906 There is no additional rationale provided for this section.

121907 **B.2.7 XSI Interprocess Communication**

121908 There are two forms of IPC supported as options in POSIX.1-2017. The traditional System V IPC
121909 routines derived from the SVID—that is, the *msg**(), *sem**(), and *shm**() interfaces—are
121910 mandatory on XSI-conformant systems. Thus, all XSI-conformant systems provide the same
121911 mechanisms for manipulating messages, shared memory, and semaphores.

121912 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems
121913 supporting the appropriate options.

121914 The application developer is presented with a choice: the System V interfaces or the POSIX
121915 interfaces (loosely derived from the Berkeley interfaces). The XSI profile prefers the System V
121916 interfaces, but the POSIX interfaces may be more suitable for realtime or other performance-
121917 sensitive applications.

121918 B.2.7.1 *IPC General Information*

121919 General information that is shared by all three mechanisms is described in this section. The
121920 common permissions mechanism is briefly introduced, describing the mode bits, and how they
121921 are used to determine whether or not a process has access to read or write/alter the appropriate
121922 instance of one of the IPC mechanisms. All other relevant information is contained in the
121923 reference pages themselves.

121924 The semaphore type of IPC allows processes to communicate through the exchange of
121925 semaphore values. A semaphore is a positive integer. Since many applications require the use of
121926 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of
121927 semaphores.

121928 Calls to support semaphores include:

121929 `semctl()`, `semget()`, `semop()`

121930 Semaphore sets are created by using the `semget()` function.

121931 The message type of IPC allows processes to communicate through the exchange of data stored
121932 in buffers. This data is transmitted between processes in discrete portions known as messages.

121933 Calls to support message queues include:

121934 `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

121935 The shared memory type of IPC allows two or more processes to share memory and
121936 consequently the data contained therein. This is done by allowing processes to set up access to a
121937 common memory address space. This sharing of memory provides a fast means of exchange of
121938 data between processes.

121939 Calls to support shared memory include:

121940 `shmctl()`, `shmdt()`, `shmget()`

121941 The `ftok()` interface is also provided.

121942 **B.2.8 Realtime**

121943 **Advisory Information**

121944 POSIX.1b contains an Informative Annex with proposed interfaces for “realtime files”. These
121945 interfaces could determine groups of the exact parameters required to do “direct I/O” or
121946 “extents”. These interfaces were objected to by a significant portion of the balloting group as too
121947 complex. A conforming application had little chance of correctly navigating the large parameter
121948 space to match its desires to the system. In addition, they only applied to a new type of file
121949 (realtime files) and they told the implementation exactly what to do as opposed to advising the
121950 implementation on application behavior and letting it optimize for the system the (portable)
121951 application was running on. For example, it was not clear how a system that had a disk array
121952 should set its parameters.

121953 There seemed to be several overall goals:

121954 Optimizing sequential access

121955 Optimizing caching behavior

121956 Optimizing I/O data transfer

121957 Preallocation

121958 The advisory interfaces, `posix_fadvise()` and `posix_madvise()`, satisfy the first two goals. The
121959 `POSIX_FADV_SEQUENTIAL` and `POSIX_MADV_SEQUENTIAL` advice tells the
121960 implementation to expect serial access. Typically the system will prefetch the next several serial
121961 accesses in order to overlap I/O. It may also free previously accessed serial data if memory is
121962 tight. If the application is not doing serial access it can use `POSIX_FADV_WILLNEED` and
121963 `POSIX_MADV_WILLNEED` to accomplish I/O overlap, as required. When the application
121964 advises `POSIX_FADV_RANDOM` or `POSIX_MADV_RANDOM` behavior, the implementation
121965 usually tries to fetch a minimum amount of data with each request and it does not expect much
121966 locality. `POSIX_FADV_DONTNEED` and `POSIX_MADV_DONTNEED` allow the system to free
121967 up caching resources as the data will not be required in the near future.

121968 `POSIX_FADV_NOREUSE` tells the system that caching the specified data is not optimal. For file
121969 I/O, the transfer should go directly to the user buffer instead of being cached internally by the

121970 implementation. To portably perform direct disk I/O on all systems, the application must
121971 perform its I/O transfers according to the following rules:

- 121972 1. The user buffer should be aligned according to the {POSIX_REC_XFER_ALIGN}
121973 *pathconf()* variable.
- 121974 2. The number of bytes transferred in an I/O operation should be a multiple of the
121975 {POSIX_ALLOC_SIZE_MIN} *pathconf()* variable.
- 121976 3. The offset into the file at the start of an I/O operation should be a multiple of the
121977 {POSIX_ALLOC_SIZE_MIN} *pathconf()* variable.
- 121978 4. The application should ensure that all threads which open a given file specify
121979 POSIX_FADV_NOREUSE to be sure that there is no unexpected interaction between
121980 threads using buffered I/O and threads using direct I/O to the same file.

121981 In some cases, a user buffer must be properly aligned in order to be transferred directly to/from
121982 the device. The {POSIX_REC_XFER_ALIGN} *pathconf()* variable tells the application the proper
121983 alignment.

121984 The preallocation goal is met by the space control function, *posix_fallocate()*. The application can
121985 use *posix_fallocate()* to guarantee no [ENOSPC] errors and to improve performance by prepaying
121986 any overhead required for block allocation.

121987 Implementations may use information conveyed by a previous *posix_fadvise()* call to influence
121988 the manner in which allocation is performed. For example, if an application did the following
121989 calls:

```
121990 fd = open("file");
121991 posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
121992 posix_fallocate(fd, len, size);
```

121993 an implementation might allocate the file contiguously on disk.

121994 Finally, the *pathconf()* variables {POSIX_REC_MIN_XFER_SIZE},
121995 {POSIX_REC_MAX_XFER_SIZE}, and {POSIX_REC_INCR_XFER_SIZE} tell the application a
121996 range of transfer sizes that are recommended for best I/O performance.

121997 Where bounded response time is required, the vendor can supply the appropriate settings of the
121998 advisories to achieve a guaranteed performance level.

121999 The interfaces meet the goals while allowing applications using regular files to take advantage
122000 of performance optimizations. The interfaces tell the implementation expected application
122001 behavior which the implementation can use to optimize performance on a particular system
122002 with a particular dynamic load.

122003 The *posix_memalign()* function was added to allow for the allocation of specifically aligned
122004 buffers; for example, for {POSIX_REC_XFER_ALIGN}.

122005 The working group also considered the alternative of adding a function which would return an
122006 aligned pointer to memory within a user-supplied buffer. This was not considered to be the best
122007 method, because it potentially wastes large amounts of memory when buffers need to be aligned
122008 on large alignment boundaries.

122009 **Message Passing**

122010 This section provides the rationale for the definition of the message passing interface in
 122011 POSIX.1-2017. This is presented in terms of the objectives, models, and requirements imposed
 122012 upon this interface.

122013 Objectives

122014 Many applications, including both realtime and database applications, require a means of
 122015 passing arbitrary amounts of data between cooperating processes comprising the overall
 122016 application on one or more processors. Many conventional interfaces for interprocess
 122017 communication are insufficient for realtime applications in that efficient and deterministic
 122018 data passing methods cannot be implemented. This has prompted the definition of
 122019 message passing interfaces providing these facilities:

122020 ‡ p~~o~~ a message queue.

122021 ‡ e~~n~~s a message to a message queue.

122022 ‡ e~~c~~rive a message from a queue, either synchronously or asynchronously.

122023 ‡ l~~t~~a message queue attributes for flow and resource control.

122024 It is assumed that an application may consist of multiple cooperating processes and that
 122025 these processes may wish to communicate and coordinate their activities. The message
 122026 passing facility described in POSIX.1-2017 allows processes to communicate through
 122027 system-wide queues. These message queues are accessed through names that may be
 122028 pathnames. A message queue can be opened for use by multiple sending and/or multiple
 122029 receiving processes.

122030 Background on Embedded Applications

122031 Interprocess communication utilizing message passing is a key facility for the construction
 122032 of deterministic, high-performance realtime applications. The facility is present in all
 122033 realtime systems and is the framework upon which the application is constructed. The
 122034 performance of the facility is usually a direct indication of the performance of the resulting
 122035 application.

122036 Realtime applications, especially for embedded systems, are typically designed around the
 122037 performance constraints imposed by the message passing mechanisms. Applications for
 122038 embedded systems are typically very tightly constrained. Application developers expect to
 122039 design and control the entire system. In order to minimize system costs, the writer will
 122040 attempt to use all resources to their utmost and minimize the requirement to add
 122041 additional memory or processors.

122042 The embedded applications usually share address spaces and only a simple message
 122043 passing mechanism is required. The application can readily access common data incurring
 122044 only mutual-exclusion overheads. The models desired are the simplest possible with the
 122045 application building higher-level facilities only when needed.

122046 Requirements

122047 The following requirements determined the features of the message passing facilities
 122048 defined in POSIX.1-2017:

122049 ‡ a~~n~~ning of Message Queues

122050 The mechanism for gaining access to a message queue is a pathname evaluated in a
 122051 context that is allowed to be a file system name space, or it can be independent of
 122052 any file system. This is a specific attempt to allow implementations based on either
 122053 method in order to address both embedded systems and to also allow

122054	implementation in larger systems.
122055	The interface of <i>mq_open()</i> is defined to allow but not require the access control and name conflicts resulting from utilizing a file system for name resolution. All required behavior is specified for the access control case. Yet a conforming implementation, such as an embedded system kernel, may define that there are no distinctions between users and may define that all processes have all access privileges.
122056	
122057	
122058	
122059	
122060	† Embedded System Naming
122061	Embedded systems need to be able to utilize independent name spaces for accessing the various system objects. They typically do not have a file system, precluding its utilization as a common name resolution mechanism. The modularity of an embedded system limits the connections between separate mechanisms that can be allowed.
122062	
122063	
122064	
122065	
122066	Embedded systems typically do not have any access protection. Since the system does not support the mixing of applications from different areas, and usually does not even have the concept of an authorization entity, access control is not useful.
122067	
122068	
122069	† Large System Naming
122070	On systems with more functionality, the name resolution must support the ability to use the file system as the name resolution mechanism/object storage medium and to have control over access to the objects. Utilizing the pathname space can result in further errors when the names conflict with other objects.
122071	
122072	
122073	
122074	† Fixed Size of Messages
122075	The interfaces impose a fixed upper bound on the size of messages that can be sent to a specific message queue. The size is set on an individual queue basis and cannot be changed dynamically.
122076	
122077	
122078	The purpose of the fixed size is to increase the ability of the system to optimize the implementation of <i>mq_send()</i> and <i>mq_receive()</i> . With fixed sizes of messages and fixed numbers of messages, specific message blocks can be pre-allocated. This eliminates a significant amount of checking for errors and boundary conditions. Additionally, an implementation can optimize data copying to maximize performance. Finally, with a restricted range of message sizes, an implementation is better able to provide deterministic operations.
122079	
122080	
122081	
122082	
122083	
122084	
122085	† Prioritization of Messages
122086	Message prioritization allows the application to determine the order in which messages are received. Prioritization of messages is a key facility that is provided by most realtime kernels and is heavily utilized by the applications. The major purpose of having priorities in message queues is to avoid priority inversions in the message system, where a high-priority message is delayed behind one or more lower-priority messages. This allows the applications to be designed so that they do not need to be interrupted in order to change the flow of control when exceptional conditions occur. The prioritization does add additional overhead to the message operations in those cases it is actually used but a clever implementation can optimize for the FIFO case to make that more efficient.
122087	
122088	
122089	
122090	
122091	
122092	
122093	
122094	
122095	
122096	† Synchronous Notification
122097	The interface supports the ability to have a task asynchronously notified of the availability of a message on the queue. The purpose of this facility is to allow the task to perform other functions and yet still be notified that a message has become
122098	
122099	

122100 available on the queue.

122101 To understand the requirement for this function, it is useful to understand two
122102 models of application design: a single task performing multiple functions and
122103 multiple tasks performing a single function. Each of these models has advantages.

122104 Asynchronous notification is required to build the model of a single task performing
122105 multiple operations. This model typically results from either the expectation that
122106 interruption is less expensive than utilizing a separate task or from the growth of the
122107 application to include additional functions.

122108 Semaphores

122109 Semaphores are a high-performance process synchronization mechanism. Semaphores are
122110 named by null-terminated strings of characters.

122111 A semaphore is created using the *sem_init()* function or the *sem_open()* function with the
122112 *O_CREAT* flag set in *oflag*.

122113 To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor
122114 for the semaphore via *fork()*.

122115 A semaphore preserves its state when the last reference is closed. For example, if a semaphore
122116 has a value of 13 when the last reference is closed, it will have a value of 13 when it is next
122117 opened.

122118 When a semaphore is created, an initial state for the semaphore has to be provided. This value is
122119 a non-negative integer. Negative values are not possible since they indicate the presence of
122120 blocked processes. The persistence of any of these objects across a system crash or a system
122121 reboot is undefined. Conforming applications must not depend on any sort of persistence across
122122 a system reboot or a system crash.

122123 Models and Requirements

122124 A realtime system requires synchronization and communication between the processes
122125 comprising the overall application. An efficient and reliable synchronization mechanism
122126 has to be provided in a realtime system that will allow more than one schedulable process
122127 mutually-exclusive access to the same resource. This synchronization mechanism has to
122128 allow for the optimal implementation of synchronization or systems implementors will
122129 define other, more cost-effective methods.

122130 At issue are the methods whereby multiple processes (tasks) can be designed and
122131 implemented to work together in order to perform a single function. This requires
122132 interprocess communication and synchronization. A semaphore mechanism is the lowest
122133 level of synchronization that can be provided by an operating system.

122134 A semaphore is defined as an object that has an integral value and a set of blocked
122135 processes associated with it. If the value is positive or zero, then the set of blocked
122136 processes is empty; otherwise, the size of the set is equal to the absolute value of the
122137 semaphore value. The value of the semaphore can be incremented or decremented by any
122138 process with access to the semaphore and must be done as an indivisible operation. When
122139 a semaphore value is less than or equal to zero, any process that attempts to lock it again
122140 will block or be informed that it is not possible to perform the operation.

122141 A semaphore may be used to guard access to any resource accessible by more than one
122142 schedulable task in the system. It is a global entity and not associated with any particular
122143 process. As such, a method of obtaining access to the semaphore has to be provided by the
122144 operating system. A process that wants access to a critical resource (section) has to wait on
122145 the semaphore that guards that resource. When the semaphore is locked on behalf of a

122146 process, it knows that it can utilize the resource without interference by any other
122147 cooperating process in the system. When the process finishes its operation on the resource,
122148 leaving it in a well-defined state, it posts the semaphore, indicating that some other
122149 process may now obtain the resource associated with that semaphore.

122150 In this section, mutexes and condition variables are specified as the synchronization
122151 mechanisms between threads.

122152 These primitives are typically used for synchronizing threads that share memory in a
122153 single process. However, this section provides an option allowing the use of these
122154 synchronization interfaces and objects between processes that share memory, regardless of
122155 the method for sharing memory.

122156 Much experience with semaphores shows that there are two distinct uses of
122157 synchronization: locking, which is typically of short duration; and waiting, which is
122158 typically of long or unbounded duration. These distinct usages map directly onto mutexes
122159 and condition variables, respectively.

122160 Semaphores are provided in POSIX.1-2017 primarily to provide a means of
122161 synchronization for processes; these processes may or may not share memory. Mutexes
122162 and condition variables are specified as synchronization mechanisms between threads;
122163 these threads always share (some) memory. Both are synchronization paradigms that have
122164 been in widespread use for a number of years. Each set of primitives is particularly well
122165 matched to certain problems.

122166 With respect to binary semaphores, experience has shown that condition variables and
122167 mutexes are easier to use for many synchronization problems than binary semaphores. The
122168 primary reason for this is the explicit appearance of a Boolean predicate that specifies
122169 when the condition wait is satisfied. This Boolean predicate terminates a loop, including
122170 the call to *pthread_cond_wait()*. As a result, extra wakeups are benign since the predicate
122171 governs whether the thread will actually proceed past the condition wait. With stateful
122172 primitives, such as binary semaphores, the wakeup in itself typically means that the wait is
122173 satisfied. The burden of ensuring correctness for such waits is thus placed on *all* signalers
122174 of the semaphore rather than on an *explicitly coded* Boolean predicate located at the
122175 condition wait. Experience has shown that the latter creates a major improvement in safety
122176 and ease-of-use.

122177 Counting semaphores are well matched to dealing with producer/consumer problems,
122178 including those that might exist between threads of different processes, or between a signal
122179 handler and a thread. In the former case, there may be little or no memory shared by the
122180 processes; in the latter case, one is not communicating between co-equal threads, but
122181 between a thread and an interrupt-like entity. It is for these reasons that POSIX.1-2017
122182 allows semaphores to be used by threads.

122183 Mutexes and condition variables have been effectively used with and without priority
122184 inheritance, priority ceiling, and other attributes to synchronize threads that share
122185 memory. The efficiency of their implementation is comparable to or better than that of
122186 other synchronization primitives that are sometimes harder to use (for example, binary
122187 semaphores). Furthermore, there is at least one known implementation of Ada tasking that
122188 uses these primitives. Mutexes and condition variables together constitute an appropriate,
122189 sufficient, and complete set of inter-thread synchronization primitives.

122190 Efficient multi-threaded applications require high-performance synchronization
122191 primitives. Considerations of efficiency and generality require a small set of primitives
122192 upon which more sophisticated synchronization functions can be built.

122193 Standardization Issues

122194 It is possible to implement very high-performance semaphores using test-and-set
 122195 instructions on shared memory locations. The library routines that implement such a high-
 122196 performance interface have to properly ensure that a `sem_wait()` or `sem_trywait()` operation
 122197 that cannot be performed will issue a blocking semaphore system call or properly report
 122198 the condition to the application. The same interface to the application program would be
 122199 provided by a high-performance implementation.

122200 *B.2.8.1 Realtime Signals*

122201 **Realtime Signals Extension**

122202 This portion of the rationale presents models, requirements, and standardization issues relevant
 122203 to the Realtime Signals Extension. This extension provides the capability required to support
 122204 reliable, deterministic, asynchronous notification of events. While a new mechanism,
 122205 unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more
 122206 efficient implementation, the application requirements for event notification can be met with a
 122207 small number of extensions to signals. Therefore, a minimal set of extensions to signals to
 122208 support the application requirements is specified.

122209 The realtime signal extensions specified in this section are used by other realtime functions
 122210 requiring asynchronous notification:

122211 Models

122212 The model supported is one of multiple cooperating processes, each of which handles
 122213 multiple asynchronous external events. Events represent occurrences that are generated as
 122214 the result of some activity in the system. Examples of occurrences that can constitute an
 122215 event include:

122216 ‡ completion of an asynchronous I/O request

122217 ‡ expiration of a POSIX.1b timer

122218 ‡ arrival of an interprocess message

122219 ‡ generation of a user-defined event

122220 Processing of these events may occur synchronously via polling for event notifications or
 122221 asynchronously via a software interrupt mechanism. Existing practice for this model is
 122222 well established for traditional proprietary realtime operating systems, realtime
 122223 executives, and realtime extended POSIX-like systems.

122224 A contrasting model is that of “cooperating sequential processes” where each process
 122225 handles a single priority of events via polling. Each process blocks while waiting for
 122226 events, and each process depends on the preemptive, priority-based process scheduling
 122227 mechanism to arbitrate between events of different priority that need to be processed
 122228 concurrently. Existing practice for this model is also well established for small realtime
 122229 executives that typically execute in an unprotected physical address space, but it is just
 122230 emerging in the context of a fuller function operating system with multiple virtual address
 122231 spaces.

122232 It could be argued that the cooperating sequential process model, and the facilities
 122233 supported by the POSIX Threads Extension obviate a software interrupt model. But, even
 122234 with the cooperating sequential process model, the need has been recognized for a
 122235 software interrupt model to handle exceptional conditions and process aborting, so the
 122236 mechanism must be supported in any case. Furthermore, it is not the purview of
 122237 POSIX.1-2017 to attempt to convince realtime practitioners that their current application

122238 models based on software interrupts are “broken” and should be replaced by the
 122239 cooperating sequential process model. Rather, it is the charter of POSIX.1-2017 to provide
 122240 standard extensions to mechanisms that support existing realtime practice.

122241 Requirements

122242 This section discusses the following realtime application requirements for asynchronous
 122243 event notification:

122244 † **Reliable** delivery of asynchronous event notification

122245 The events notification mechanism guarantees delivery of an event notification.
 122246 Asynchronous operations (such as asynchronous I/O and timers) that complete
 122247 significantly after they are invoked have to guarantee that delivery of the event
 122248 notification can occur at the time of completion.

122249 † **Prioritized** handling of asynchronous event notifications

122250 The events notification mechanism supports the assigning of a user function as an
 122251 event notification handler. Furthermore, the mechanism supports the preemption of
 122252 an event handler function by a higher priority event notification and supports the
 122253 selection of the highest priority pending event notification when multiple
 122254 notifications (of different priority) are pending simultaneously.

122255 The model here is based on hardware interrupts. Asynchronous event handling
 122256 allows the application to ensure that time-critical events are immediately processed
 122257 when delivered, without the indeterminism of being at a random location within a
 122258 polling loop. Use of handler priority allows the specification of how handlers are
 122259 interrupted by other higher priority handlers.

122260 † **Identification** between multiple occurrences of event notifications of the same type

122261 The events notification mechanism passes an application-defined value to the event
 122262 handler function. This value can be used for a variety of purposes, such as enabling
 122263 the application to identify which of several possible events of the same type (for
 122264 example, timer expirations) has occurred.

122265 † **Polled** reception of asynchronous event notifications

122266 The events notification mechanism supports blocking and non-blocking polls for
 122267 asynchronous event notification.

122268 The polled mode of operation is often preferred over the interrupt mode by those
 122269 practitioners accustomed to this model. Providing support for this model facilitates
 122270 the porting of applications based on this model to POSIX.1b conforming systems.

122271 † **Deterministic** response to asynchronous event notifications

122272 The events notification mechanism does not preclude implementations that provide
 122273 deterministic event dispatch latency and minimizes the number of system calls
 122274 needed to use the event facilities during realtime processing.

122275 Rationale for Extension

122276 POSIX.1 signals have many of the characteristics necessary to support the asynchronous
 122277 handling of event notifications, and the Realtime Signals Extension addresses the
 122278 following deficiencies in the POSIX.1 signal mechanism:

122279 † **Signals** do not support reliable delivery of event notification. Subsequent
 122280 occurrences of a pending signal are not guaranteed to be delivered.

122281 signals do not support prioritized delivery of event notifications. The order of signal
 122282 delivery when multiple unblocked signals are pending is undefined.

122283 signals do not support the differentiation between multiple signals of the same type.

122284 B.2.8.2 Asynchronous I/O

122285 Many applications need to interact with the I/O subsystem in an asynchronous manner. The
 122286 asynchronous I/O mechanism provides the ability to overlap application processing and I/O
 122287 operations initiated by the application. The asynchronous I/O mechanism allows a single
 122288 process to perform I/O simultaneously to a single file multiple times or to multiple files
 122289 multiple times.

122290 Overview

122291 Asynchronous I/O operations proceed in logical parallel with the processing done by the
 122292 application after the asynchronous I/O has been initiated. Other than this difference,
 122293 asynchronous I/O behaves similarly to normal I/O using *read()*, *write()*, *lseek()*, and *fsync()*.
 122294 The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to
 122295 perform atomically the implied *lseek()* operation, if any, and then the requested I/O operation
 122296 (either *read()*, *write()*, or *fsync()*). There is no seek implied with a call to *aio_fsync()*. Concurrent
 122297 asynchronous operations and synchronous operations applied to the same file update the file as
 122298 if the I/O operations had proceeded serially.

122299 When asynchronous I/O completes, a signal can be delivered to the application to indicate the
 122300 completion of the I/O. This signal can be used to indicate that buffers and control blocks used
 122301 for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous
 122302 operation and may be turned off on a per-operation basis by the application. Signals may also be
 122303 synchronously polled using *aio_suspend()*, *sigtimedwait()*, or *sigwaitinfo()*.

122304 Normal I/O has a return value and an error status associated with it. Asynchronous I/O
 122305 returns a value and an error status when the operation is first submitted, but that only relates to
 122306 whether the operation was successfully queued up for servicing. The I/O operation itself also
 122307 has a return status and an error value. To allow the application to retrieve the return status and
 122308 the error value, functions are provided that, given the address of an asynchronous I/O control
 122309 block, yield the return and error status associated with the operation. Until an asynchronous I/O
 122310 operation is done, its error status is [EINPROGRESS]. Thus, an application can poll for
 122311 completion of an asynchronous I/O operation by waiting for the error status to become equal to
 122312 a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is
 122313 undefined so long as the error status is equal to [EINPROGRESS].

122314 Storage for asynchronous operation return and error status may be limited. Submission of
 122315 asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves
 122316 the return status of a given asynchronous operation, therefore, any system-maintained storage
 122317 used for this status and the error status may be reclaimed for use by other asynchronous
 122318 operations.

122319 Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b
 122320 synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein;
 122321 however, the asynchronous operation I/O in this case is not completed until the I/O has reached
 122322 either the state of synchronized I/O data integrity completion or synchronized I/O file integrity
 122323 completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

122324 **Models**

122325 Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition
122326 model, and a model of the use of asynchronous I/O in supercomputing applications.

122327 Journalization Model

122328 Many realtime applications perform low-priority journalizing functions. Journalizing
122329 requires that logging records be queued for output without blocking the initiating process.

122330 Data Acquisition Model

122331 A data acquisition process may also serve as a model. The process has two or more
122332 channels delivering intermittent data that must be read within a certain time. The process
122333 issues one asynchronous read on each channel. When one of the channels needs data
122334 collection, the process reads the data and posts it through an asynchronous write to
122335 secondary memory for future processing.

122336 Supercomputing Model

122337 The supercomputing community has used asynchronous I/O much like that specified in
122338 POSIX.1 for many years. This community requires the ability to perform multiple I/O
122339 operations to multiple devices with a minimal number of entries to "the system"; each
122340 entry to "the system" provokes a major delay in operations when compared to the normal
122341 progress made by the application. This existing practice motivated the use of combined
122342 *lseek()* and *read()* or *write()* calls, as well as the *lio_listio()* call. Another common practice is
122343 to disable signal notification for I/O completion, and simply poll for I/O completion at
122344 some interval by which the I/O should be completed. Likewise, interfaces like *aio_cancel()*
122345 have been in successful commercial use for many years. Note also that an underlying
122346 implementation of asynchronous I/O will require the ability, at least internally, to cancel
122347 outstanding asynchronous I/O, at least when the process exits. (Consider an asynchronous
122348 read from a terminal, when the process intends to exit immediately.)

122349 **Requirements**

122350 Asynchronous input and output for realtime implementations have these requirements:

122351 The ability to queue multiple asynchronous read and write operations to a single open
122352 instance. Both sequential and random access should be supported.

122353 The ability to queue asynchronous read and write operations to multiple open instances.

122354 The ability to obtain completion status information by polling and/or asynchronous event
122355 notification.

122356 Asynchronous event notification on asynchronous I/O completion is optional.

122357 It has to be possible for the application to associate the event with the *aiocbp* for the
122358 operation that generated the event.

122359 The ability to cancel queued requests.

122360 The ability to wait upon asynchronous I/O completion in conjunction with other types of
122361 events.

122362 The ability to accept an *aio_read()* and an *aio_cancel()* for a device that accepts a *read()*, and
122363 the ability to accept an *aio_write()* and an *aio_cancel()* for a device that accepts a *write()*.
122364 This does not imply that the operation is asynchronous.

122365 **Standardization Issues**

122366 The following issues are addressed by the standardization of asynchronous I/O:

122367 Rationale for New Interface

122368 Non-blocking I/O does not satisfy the needs of either realtime or high-performance
 122369 computing models; these models require that a process overlap program execution and
 122370 I/O processing. Realtime applications will often make use of direct I/O to or from the
 122371 address space of the process, or require synchronized (unbuffered) I/O; they also require
 122372 the ability to overlap this I/O with other computation. In addition, asynchronous I/O
 122373 allows an application to keep a device busy at all times, possibly achieving greater
 122374 throughput. Supercomputing and database architectures will often have specialized
 122375 hardware that can provide true asynchrony underlying the logical asynchrony provided
 122376 by this interface. In addition, asynchronous I/O should be supported by all types of files
 122377 and devices in the same manner.

122378 Effect of Buffering

122379 If asynchronous I/O is performed on a file that is buffered prior to being actually written
 122380 to the device, it is possible that asynchronous I/O will offer no performance advantage
 122381 over normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away
 122382 from the running process and the I/O will occur at interrupt time. This potential lack of
 122383 gain in performance in no way obviates the need for asynchronous I/O by realtime
 122384 applications, which very often will use specialized hardware support, multiple processors,
 122385 and/or unbuffered, synchronized I/O.

122386 *B.2.8.3 Memory Management*

122387 All memory management and shared memory definitions are located in the `<sys/mman.h>`
 122388 header. This is for alignment with historical practice.

122389 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/7 is applied, correcting the shading and
 122390 margin markers in the introduction to Section 2.8.3.1.

122391 **Memory Locking Functions**

122392 This portion of the rationale presents models, requirements, and standardization issues relevant
 122393 to process memory locking.

122394 Models

122395 Realtime systems that conform to POSIX.1-2017 are expected (and desired) to be supported
 122396 on systems with demand-paged virtual memory management, non-paged swapping
 122397 memory management, and physical memory systems with no memory management
 122398 hardware. The general case, however, is the demand-paged, virtual memory system with
 122399 each POSIX process running in a virtual address space. Note that this includes
 122400 architectures where each process resides in its own virtual address space and architectures
 122401 where the address space of each process is only a portion of a larger global virtual address
 122402 space.

122403 The concept of memory locking is introduced to eliminate the indeterminacy introduced
 122404 by paging and swapping, and to support an upper bound on the time required to access
 122405 the memory mapped into the address space of a process. Ideally, this upper bound will be
 122406 the same as the time required for the processor to access “main memory”, including any
 122407 address translation and cache miss overheads. But some implementations—primarily on
 122408 mainframes—will not actually force locked pages to be loaded and held resident in main
 122409 memory. Rather, they will handle locked pages so that accesses to these pages will meet the

122410 performance metrics for locked process memory in the implementation. Also, although it
122411 is not, for example, the intention that this interface, as specified, be used to lock process
122412 memory into “cache”, it is conceivable that an implementation could support a large static
122413 RAM memory and define this as “main memory” and use a large[r] dynamic RAM as
122414 “backing store”. These interfaces could then be interpreted as supporting the locking of
122415 process memory into the static RAM. Support for multiple levels of backing store would
122416 require extensions to these interfaces.

122417 Implementations may also use memory locking to guarantee a fixed translation between
122418 virtual and physical addresses where such is beneficial to improving determinacy for
122419 direct-to/from-process input/output. POSIX.1-2017 does not guarantee to the application
122420 that the virtual-to-physical address translations, if such exist, are fixed, because such
122421 behavior would not be implementable on all architectures on which implementations of
122422 POSIX.1-2017 are expected. But POSIX.1-2017 does mandate that an implementation
122423 define, for the benefit of potential users, whether or not locking guarantees fixed
122424 translations.

122425 Memory locking is defined with respect to the address space of a process. Only the pages
122426 mapped into the address space of a process may be locked by the process, and when the
122427 pages are no longer mapped into the address space—for whatever reason †the locks
122428 established with respect to that address space are removed. Shared memory areas warrant
122429 special mention, as they may be mapped into more than one address space or mapped
122430 more than once into the address space of a process; locks may be established on pages
122431 within these areas with respect to several of these mappings. In such a case, the lock state
122432 of the underlying physical pages is the logical OR of the lock state with respect to each of
122433 the mappings. Only when all such locks have been removed are the shared pages
122434 considered unlocked.

122435 In recognition of the page granularity of Memory Management Units (MMU), and in order
122436 to support locking of ranges of address space, memory locking is defined in terms of
122437 “page” granularity. That is, for the interfaces that support an address and size specification
122438 for the region to be locked, the address must be on a page boundary, and all pages mapped
122439 by the specified range are locked, if valid. This means that the length is implicitly rounded
122440 up to a multiple of the page size. The page size is implementation-defined and is available
122441 to applications as a compile-time symbolic constant or at runtime via *sysconf()*.

122442 A “real memory” POSIX.1b implementation that has no MMU could elect not to support
122443 these interfaces, returning [ENOSYS]. But an application could easily interpret this as
122444 meaning that the implementation would unconditionally page or swap the application
122445 when such is not the case. It is the intention of POSIX.1-2017 that such a system could
122446 define these interfaces as “NO-OPs”, returning success without actually performing any
122447 function except for mandated argument checking.

122448 Requirements

122449 For realtime applications, memory locking is generally considered to be required as part of
122450 application initialization. This locking is performed after an application has been loaded
122451 (that is, *exec'd*) and the program remains locked for its entire lifetime. But to support
122452 applications that undergo major mode changes where, in one mode, locking is required,
122453 but in another it is not, the specified interfaces allow repeated locking and unlocking of
122454 memory within the lifetime of a process.

122455 When a realtime application locks its address space, it should not be necessary for the
122456 application to then “touch” all of the pages in the address space to guarantee that they are
122457 resident or else suffer potential paging delays the first time the page is referenced. Thus,
122458 POSIX.1-2017 requires that the pages locked by the specified interfaces be resident when

122459 the locking functions return successfully.

122460 Many architectures support system-managed stacks that grow automatically when the
122461 current extent of the stack is exceeded. A realtime application has a requirement to be able
122462 to “preallocate” sufficient stack space and lock it down so that it will not suffer page faults
122463 to grow the stack during critical realtime operation. There was no consensus on a portable
122464 way to specify how much stack space is needed, so POSIX.1-2017 supports no specific
122465 interface for preallocating stack space. But an application can portably lock down a specific
122466 amount of stack space by specifying `MCL_FUTURE` in a call to `mlockall()` and then calling
122467 a dummy function that declares an automatic array of the desired size.

122468 Memory locking for realtime applications is also generally considered to be an “all or
122469 nothing” proposition. That is, the entire process, or none, is locked down. But, for
122470 applications that have well-defined sections that need to be locked and others that do not,
122471 POSIX.1-2017 supports an optional set of interfaces to lock or unlock a range of process
122472 addresses. Reasons for locking down a specific range include:

122473 ‡ `rt_async` Asynchronous event handler function that must respond to external events in a
122474 deterministic manner such that page faults cannot be tolerated

122475 ‡ `rt_io` Input/output “buffer” area that is the target for direct-to-process I/O, and the
122476 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

122477 Finally, locking is generally viewed as an “application-wide” function. That is, the
122478 application is globally aware of which regions are locked and which are not over time. This
122479 is in contrast to a function that is used temporarily within a “third party” library routine
122480 whose function is unknown to the application, and therefore must have no “side-effects”.
122481 The specified interfaces, therefore, do not support “lock stacking” or “lock nesting” within
122482 a process. But, for pages that are shared between processes or mapped more than once
122483 into a process address space, “lock stacking” is essentially mandated by the requirement
122484 that unlocking of pages that are mapped by more than one process or more than once by
122485 the same process does not affect locks established on the other mappings.

122486 There was some support for “lock stacking” so that locking could be transparently used in
122487 functions or opaque modules. But the consensus was not to burden all implementations
122488 with lock stacking (and reference counting), and an implementation option was proposed.
122489 There were strong objections to the option because applications would have to support
122490 both options in order to remain portable. The consensus was to eliminate lock stacking
122491 altogether, primarily through overwhelming support for the System V “`m[un]lock[all]`”
122492 interface on which POSIX.1-2017 is now based.

122493 Locks are not inherited across `fork()`s because some implementations implement `fork()` by
122494 creating new address spaces for the child. In such an implementation, requiring locks to be
122495 inherited would lead to new situations in which a fork would fail due to the inability of
122496 the system to lock sufficient memory to lock both the parent and the child. The consensus
122497 was that there was no benefit to such inheritance. Note that this does not mean that locks
122498 are removed when, for instance, a thread is created in the same address space.

122499 Similarly, locks are not inherited across `exec` because some implementations implement `exec`
122500 by unmapping all of the pages in the address space (which, by definition, removes the
122501 locks on these pages), and maps in pages of the `exec`’d image. In such an implementation,
122502 requiring locks to be inherited would lead to new situations in which `exec` would fail.
122503 Reporting this failure would be very cumbersome to detect in time to report to the calling
122504 process, and no appropriate mechanism exists for informing the `exec`’d process of its status.

122505 It was determined that, if the newly loaded application required locking, it was the
122506 responsibility of that application to establish the locks. This is also in keeping with the

122507 general view that it is the responsibility of the application to be aware of all locks that are
122508 established.

122509 There was one request to allow (not mandate) locks to be inherited across *fork()*, and a
122510 request for a flag, *MCL_INHERIT*, that would specify inheritance of memory locks across
122511 *execs*. Given the difficulties raised by this and the general lack of support for the feature in
122512 POSIX.1-2017, it was not added. POSIX.1-2017 does not preclude an implementation from
122513 providing this feature for administrative purposes, such as a “run” command that will
122514 lock down and execute a specified application. Additionally, the rationale for the objection
122515 equated *fork()* with creating a thread in the address space. POSIX.1-2017 does not mandate
122516 releasing locks when creating additional threads in an existing process.

122517 Standardization Issues

122518 One goal of POSIX.1-2017 is to define a set of primitives that provide the necessary
122519 functionality for realtime applications, with consideration for the needs of other
122520 application domains where such were identified, which is based to the extent possible on
122521 existing industry practice.

122522 The Memory Locking option is required by many realtime applications to tune
122523 performance. Such a facility is accomplished by placing constraints on the virtual memory
122524 system to limit paging of time of the process or of critical sections of the process. This
122525 facility should not be used by most non-realtime applications.

122526 Optional features provided in POSIX.1-2017 allow applications to lock selected address
122527 ranges with the caveat that the process is responsible for being aware of the page
122528 granularity of locking and the unnested nature of the locks.

122529 Mapped Files Functions

122530 The memory mapped files functionality provides a mechanism that allows a process to access
122531 files by directly incorporating file data into its address space. Once a file is “mapped” into a
122532 process address space, the data can be manipulated by instructions as memory. The use of
122533 mapped files can significantly reduce I/O data movement since file data does not have to be
122534 copied into process data buffers as in *read()* and *write()*. If more than one process maps a file, its
122535 contents are shared among them. This provides a low overhead mechanism by which processes
122536 can synchronize and communicate.

122537 Historical Perspective

122538 Realtime applications have historically been implemented using a collection of cooperating
122539 processes or tasks. In early systems, these processes ran on bare hardware (that is, without
122540 an operating system) with no memory relocation or protection. The application paradigms
122541 that arose from this environment involve the sharing of data between the processes.

122542 When realtime systems were implemented on top of vendor-supplied operating systems,
122543 the paradigm or performance benefits of direct access to data by multiple processes was
122544 still deemed necessary. As a result, operating systems that claim to support realtime
122545 applications must support the shared memory paradigm.

122546 Additionally, a number of realtime systems provide the ability to map specific sections of
122547 the physical address space into the address space of a process. This ability is required if an
122548 application is to obtain direct access to memory locations that have specific properties (for
122549 example, refresh buffers or display devices, dual ported memory locations, DMA target
122550 locations). The use of this ability is common enough to warrant some degree of
122551 standardization of its interface. This ability overlaps the general paradigm of shared
122552 memory in that, in both instances, common global objects are made addressable by
122553 individual processes or tasks.

122554 Finally, a number of systems also provide the ability to map process addresses to files. This
 122555 provides both a general means of sharing persistent objects, and using files in a manner
 122556 that optimizes memory and swapping space usage.

122557 Simple shared memory is clearly a special case of the more general file mapping capability.
 122558 In addition, there is relatively widespread agreement and implementation of the file
 122559 mapping interface. In these systems, many different types of objects can be mapped (for
 122560 example, files, memory, devices, and so on) using the same mapping interfaces. This
 122561 approach both minimizes interface proliferation and maximizes the generality of programs
 122562 using the mapping interfaces.

122563 Memory Mapped Files Usage

122564 A memory object can be concurrently mapped into the address space of one or more
 122565 processes. The *mmap()* and *munmap()* functions allow a process to manipulate their
 122566 address space by mapping portions of memory objects into it and removing them from it.
 122567 When multiple processes map the same memory object, they can share access to the
 122568 underlying data. Implementations may restrict the size and alignment of mappings to be
 122569 on *page*-size boundaries. The page size, in bytes, is the value of the system-configurable
 122570 variable {PAGESIZE}, typically accessed by calling *sysconf()* with a *name* argument of
 122571 *_SC_PAGESIZE*. If an implementation has no restrictions on size or alignment, it may
 122572 specify a 1-byte page size.

122573 To map memory, a process first opens a memory object. The *ftruncate()* function can be
 122574 used to contract or extend the size of the memory object even when the object is currently
 122575 mapped. If the memory object is extended, the contents of the extended areas are zeros.

122576 After opening a memory object, the application maps the object into its address space
 122577 using the *mmap()* function call. Once a mapping has been established, it remains mapped
 122578 until unmapped with *munmap()*, even if the memory object is closed. The *mprotect()*
 122579 function can be used to change the memory protections initially established by *mmap()*.

122580 A *close()* of the file descriptor, while invalidating the file descriptor itself, does not unmap
 122581 any mappings established for the memory object. The address space, including all mapped
 122582 regions, is inherited on *fork()*. The entire address space is unmapped on process
 122583 termination or by successful calls to any of the *exec* family of functions.

122584 The *msync()* function is used to force mapped file data to permanent storage.

122585 Effects on Other Functions

122586 With memory mapped files, the operation of the *open()*, *creat()*, and *unlink()* functions are
 122587 a natural result of using the file system name space to map the global names for memory
 122588 objects.

122589 The *ftruncate()* function can be used to set the length of a sharable memory object.

122590 The meaning of *stat()* fields other than the size and protection information is undefined on
 122591 implementations where memory objects are not implemented using regular files. When
 122592 regular files are used, the times reflect when the implementation updated the file image of
 122593 the data, not when a process updated the data in memory.

122594 The operations of *fdopen()*, *write()*, *read()*, and *lseek()* were made unspecified for objects
 122595 opened with *shm_open()*, so that implementations that did not implement memory objects
 122596 as regular files would not have to support the operation of these functions on shared
 122597 memory objects.

122598 The behavior of memory objects with respect to *close()*, *dup()*, *dup2()*, *open()*, *close()*,
 122599 *fork()*, *_exit()*, and the *exec* family of functions is the same as the behavior of the existing

- 122600 practice of the *mmap()* function.
- 122601 A memory object can still be referenced after a close. That is, any mappings made to the
122602 file are still in effect, and reads and writes that are made to those mappings are still valid
122603 and are shared with other processes that have the same mapping. Likewise, the memory
122604 object can still be used if any references remain after its name(s) have been deleted. Any
122605 references that remain after a close must not appear to the application as file descriptors.
- 122606 This is existing practice for *mmap()* and *close()*. In addition, there are already mappings
122607 present (text, data, stack) that do not have open file descriptors. The text mapping in
122608 particular is considered a reference to the file containing the text. The desire was to treat all
122609 mappings by the process uniformly. Also, many modern implementations use *mmap()* to
122610 implement shared libraries, and it would not be desirable to keep file descriptors for each
122611 of the many libraries an application can use. It was felt there were many other existing
122612 programs that used this behavior to free a file descriptor, and thus POSIX.1-2017 could not
122613 forbid it and still claim to be using existing practice.
- 122614 For implementations that implement memory objects using memory only, memory objects
122615 will retain the memory allocated to the file after the last close and will use that same
122616 memory on the next open. Note that closing the memory object is not the same as deleting
122617 the name, since the memory object is still defined in the memory object name space.
- 122618 The locks of *fcntl()* do not block any read or write operation, including read or write access
122619 to shared memory or mapped files. In addition, implementations that only support shared
122620 memory objects should not be required to implement record locks. The reference to *fcntl()*
122621 is added to make this point explicitly. The other *fcntl()* commands are useful with shared
122622 memory objects.
- 122623 The size of pages that mapping hardware may be able to support may be a configurable
122624 value, or it may change based on hardware implementations. The addition of the
122625 *_SC_PAGESIZE* parameter to the *sysconf()* function is provided for determining the
122626 mapping page size at runtime.

122627 Shared Memory Functions

- 122628 Implementations may support the Shared Memory Objects option independently of memory
122629 mapped files. Shared memory objects are named regions of storage that may be independent of
122630 the file system and can be mapped into the address space of one or more processes to allow
122631 them to share the associated memory.
- 122632 Requirements
- 122633 Shared memory is used to share data among several processes, each potentially running at
122634 different priority levels, responding to different inputs, or performing separate tasks.
122635 Shared memory is not just simply providing common access to data, it is providing the
122636 fastest possible communication between the processes. With one memory write operation,
122637 a process can pass information to as many processes as have the memory region mapped.
- 122638 As a result, shared memory provides a mechanism that can be used for all other
122639 interprocess communication facilities. It may also be used by an application for
122640 implementing more sophisticated mechanisms than semaphores and message queues.
- 122641 The need for a shared memory interface is obvious for virtual memory systems, where the
122642 operating system is directly preventing processes from accessing each other's data.
122643 However, in unprotected systems, such as those found in some embedded controllers, a
122644 shared memory interface is needed to provide a portable mechanism to allocate a region of
122645 memory to be shared and then to communicate the address of that region to other
122646 processes.

122647 This, then, provides the minimum functionality that a shared memory interface must have
 122648 in order to support realtime applications: to allocate and name an object to be mapped into
 122649 memory for potential sharing (*open()* or *shm_open()*), and to make the memory object
 122650 available within the address space of a process (*mmap()*). To complete the interface, a
 122651 mechanism to release the claim of a process on a shared memory object (*munmap()*) is also
 122652 needed, as well as a mechanism for deleting the name of a sharable object that was
 122653 previously created (*unlink()* or *shm_unlink()*).

122654 After a mapping has been established, an implementation should not have to provide
 122655 services to maintain that mapping. All memory writes into that area will appear
 122656 immediately in the memory mapping of that region by any other processes.

122657 Thus, requirements include:

122658 ‡ support creation of sharable memory objects and the mapping of these objects into
 122659 the address space of a process.

122660 ‡ sharable memory objects should be accessed by global names accessible from all
 122661 processes.

122662 ‡ support the mapping of specific sections of physical address space (such as a
 122663 memory mapped device) into the address space of a process. This should not be
 122664 done by the process specifying the actual address, but again by an implementation-
 122665 defined global name (such as a special device name) dedicated to this purpose.

122666 ‡ support the mapping of discrete portions of these memory objects.

122667 ‡ support for minimum hardware configurations that contain no physical media on
 122668 which to store shared memory contents permanently.

122669 ‡ the ability to preallocate the entire shared memory region so that minimum
 122670 hardware configurations without virtual memory support can guarantee contiguous
 122671 space.

122672 ‡ the maximizing of performance by not requiring functionality that would require
 122673 implementation interaction above creating the shared memory area and returning
 122674 the mapping.

122675 Note that the above requirements do not preclude:

122676 ‡ the sharable memory object from being implemented using actual files on an actual
 122677 file system.

122678 ‡ the global name that is accessible from all processes being restricted to a file system
 122679 area that is dedicated to handling shared memory.

122680 ‡ the implementation not providing implementation-defined global names for the
 122681 purpose of physical address mapping.

122682 Shared Memory Objects Usage

122683 If the Shared Memory Objects option is supported, a shared memory object may be
 122684 created, or opened if it already exists, with the *shm_open()* function. If the shared memory
 122685 object is created, it has a length of zero. The *ftruncate()* function can be used to set the size
 122686 of the shared memory object after creation. The *shm_unlink()* function removes the name
 122687 for a shared memory object created by *shm_open()*.

122688 Shared Memory Overview

122689 The shared memory facility defined by POSIX.1-2017 usually results in memory locations
 122690 being added to the address space of the process. The implementation returns the address

122691 of the new space to the application by means of a pointer. This works well in languages
122692 like C. However, in languages without pointer types it will not work. In the bindings for
122693 such a language, either a special COMMON section will need to be defined (which is
122694 unlikely), or the binding will have to allow existing structures to be mapped. The
122695 implementation will likely have to place restrictions on the size and alignment of such
122696 structures or will have to map a suitable region of the address space of the process into the
122697 memory object, and thus into other processes. These are issues for that particular language
122698 binding. For POSIX.1-2017, however, the practice will not be forbidden, merely undefined.

122699 Two potentially different name spaces are used for naming objects that may be mapped
122700 into process address spaces. When using memory mapped files, files may be accessed via
122701 *open()*. When the Shared Memory Objects option is supported, sharable memory objects
122702 that might not be files may be accessed via the *shm_open()* function. These operations are
122703 not mutually-exclusive.

122704 Some implementations supporting the Shared Memory Objects option may choose to
122705 implement the shared memory object name space as part of the file system name space.
122706 There are several reasons for this:

122707 ‡ allows applications to prevent name conflicts by use of the directory structure.

122708 ‡ uses an existing mechanism for accessing global objects and prevents the creation
122709 of a new mechanism for naming global objects.

122710 In such implementations, memory objects can be implemented using regular files, if that is
122711 what the implementation chooses. The *shm_open()* function can be implemented as an
122712 *open()* call in a fixed directory with the O_CLOEXEC flag set. The *shm_unlink()* function
122713 can be implemented as an *unlink()* call.

122714 On the other hand, it is also expected that small embedded systems that support the
122715 Shared Memory Objects option may wish to implement shared memory without having
122716 any file systems present. In this case, the implementations may choose to use a simple
122717 string valued name space for shared memory regions. The *shm_open()* function permits
122718 either type of implementation.

122719 Some implementations have hardware that supports protection of mapped data from
122720 certain classes of access and some do not. Systems that supply this functionality support
122721 the memory protection functionality.

122722 Some implementations restrict size, alignment, and protections to be on *page-size*
122723 boundaries. If an implementation has no restrictions on size or alignment, it may specify a
122724 1-byte page size. Applications on implementations that do support larger pages must be
122725 cognizant of the page size since this is the alignment and protection boundary.

122726 Simple embedded implementations may have a 1-byte page size and only support the
122727 Shared Memory Objects option. This provides simple shared memory between processes
122728 without requiring mapping hardware.

122729 POSIX.1-2017 specifically allows a memory object to remain referenced after a close
122730 because that is existing practice for the *mmap()* function.

122731 **Typed Memory Functions**

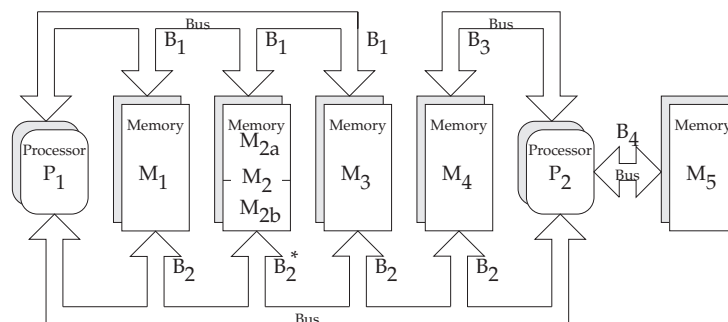
122732 Implementations may support the Typed Memory Objects option without supporting either the
 122733 Shared Memory option or memory mapped files. Types memory objects are pools of specialized
 122734 storage, different from the main memory resource normally used by a processor to hold code
 122735 and data, that can be mapped into the address space of one or more processes.

122736 **Model**

122737 Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and
 122738 desired) to be supported on systems with more than one type or pool of memory (for
 122739 example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory
 122740 may be accessible by one or more processors via one or more buses (ports). Memory
 122741 mapped files, shared memory objects, and the language-specific storage allocation
 122742 operators (*malloc()* for the ISO C standard, *new* for ISO Ada) fail to provide application
 122743 program interfaces versatile enough to allow applications to control their utilization of
 122744 such diverse memory resources. The typed memory interfaces *posix_typed_mem_open()*,
 122745 *posix_mem_offset()*, *posix_typed_mem_get_info()*, *mmap()*, and *munmap()* defined herein
 122746 support the model of typed memory described below.

122747 For purposes of this model, a system comprises several processors (for example, P_1 and
 122748 P_2), several physical memory pools (for example, $M_1, M_2, M_{2a}, M_{2b}, M_3, M_4,$ and M_5), and
 122749 several buses or "ports" (for example, $B_1, B_2, B_3,$ and B_4) interconnecting the various
 122750 processors and memory pools in some system-specific way. Notice that some memory
 122751 pools may be contained in others (for example, M_{2a} and M_{2b} are contained in M_2).

122752 **Figure B-1** shows an example of such a model. In a system like this, an application should
 122753 be able to perform the following operations:



- * All addresses in pool M_2 (comprising pools M_{2a} and M_{2b}) accessible via port B_1 .
- Addresses in pool M_{2b} are also accessible via port B_2 .
- Addresses in pool M_{2a} are *not* accessible via port B_2 .

122754 **Figure B-1** Example of a System with Typed Memory122755 **Typed Memory Allocation**

122756 An application should be able to allocate memory dynamically from the desired pool
 122757 using the desired bus, and map it into the address space of a process. For example,
 122758 processor P_1 can allocate some portion of memory pool M_1 through port B_1 , treating
 122759 all unmapped subareas of M_1 as a heap-storage resource from which memory may be
 122760 allocated. This portion of memory is mapped into address space of the process, and
 122761 subsequently deallocated when unmapped from all processes.

122762	‡ sharing the Same Storage Region from Different Buses
122763	An application process with a mapped region of storage that is accessed from one bus should be able to map that same storage area at another address (subject to page size restrictions detailed in <i>mmap()</i>), to allow it to be accessed from another bus. For example, processor P_1 may wish to access the same region of memory pool M_{2b} both through ports B_1 and B_2 .
122764	
122765	
122766	
122767	
122768	‡ sharing Typed Memory Regions
122769	Several application processes running on the same or different processors may wish to share a particular region of a typed memory pool. Each process or processor may wish to access this region through different buses. For example, processor P_1 may want to share a region of memory pool M_4 with processor P_2 , and they may be required to use buses B_2 and B_3 , respectively, to minimize bus contention. A problem arises here when a process allocates and maps a portion of fragmented memory and then wants to share this region of memory with another process, either in the same processor or different processors. The solution adopted is to allow the first process to find out the memory map (offsets and lengths) of all the different fragments of memory that were mapped into its address space, by repeatedly calling <i>posix_mem_offset()</i> . Then, this process can pass the offsets and lengths obtained to the second process, which can then map the same memory fragments into its address space.
122770	
122771	
122772	
122773	
122774	
122775	
122776	
122777	
122778	
122779	
122780	
122781	
122782	‡ contiguous Allocation
122783	The problem of finding the memory map of the different fragments of the memory pool that were mapped into logically contiguous addresses of a given process can be solved by requesting contiguous allocation. For example, a process in P_1 can allocate 10 Kbytes of physically contiguous memory from M_3-B_1 , and obtain the offset (within pool M_3) of this block of memory. Then, it can pass this offset (and the length) to a process in P_2 using some interprocess communication mechanism. The second process can map the same block of memory by using the offset transferred and specifying M_3-B_2 .
122784	
122785	
122786	
122787	
122788	
122789	
122790	
122791	‡ non Allocated Mapping
122792	Any subarea of a memory pool that is mapped to a process, either as the result of an allocation request or an explicit mapping, is normally unavailable for allocation. Special processes such as debuggers, however, may need to map large areas of a typed memory pool, yet leave those areas available for allocation.
122793	
122794	
122795	
122796	Typed memory allocation and mapping has to coexist with storage allocation operators like <i>malloc()</i> , but systems are free to choose how to implement this coexistence. For example, it may be system configuration-dependent if all available system memory is made part of one of the typed memory pools or if some part will be restricted to conventional allocation operators. Equally system configuration-dependent may be the availability of operators like <i>malloc()</i> to allocate storage from certain typed memory pools. It is not excluded to configure a system such that a given named pool, P_1 , is in turn split into non-overlapping named subpools. For example, M_1-B_1 , M_2-B_1 , and M_3-B_1 could also be accessed as one common pool $M_{123}-B_1$. A call to <i>malloc()</i> on P_1 could work on such a larger pool while full optimization of memory usage by P_1 would require typed memory allocation at the subpool level.
122797	
122798	
122799	
122800	
122801	
122802	
122803	
122804	
122805	
122806	
122807	Existing Practice
122808	OS-9 provides for the naming (numbering) and prioritization of memory types by a system administrator. It then provides APIs to request memory allocation of typed (colored)
122809	

122810 memory by number, and to generate a bus address from a mapped memory address
 122811 (translate). When requesting colored memory, the user can specify type 0 to signify
 122812 allocation from the first available type in priority order.

122813 HP-RT presents interfaces to map different kinds of storage regions that are visible through
 122814 a VME bus, although it does not provide allocation operations. It also provides functions
 122815 to perform address translation between VME addresses and virtual addresses. It represents
 122816 a VME-bus unique solution to the general problem.

122817 The PSOS approach is similar (that is, based on a pre-established mapping of bus address
 122818 ranges to specific memories) with a concept of segments and regions (regions dynamically
 122819 allocated from a heap which is a special segment). Therefore, PSOS does not fully address
 122820 the general allocation problem either. PSOS does not have a "process"-based model, but
 122821 more of a "thread"-only-based model of multi-tasking. So mapping to a process address
 122822 space is not an issue.

122823 QNX uses the System V approach of opening specially named devices (shared memory
 122824 segments) and using *mmap()* to then gain access from the process. They do not address
 122825 allocation directly, but once typed shared memory can be mapped, an "allocation
 122826 manager" process could be written to handle requests for allocation.

122827 The System V approach also included allocation, implemented by opening yet other
 122828 special "devices" which allocate, rather than appearing as a whole memory object.

122829 The Orkid realtime kernel interface definition has operations to manage memory "regions"
 122830 and "pools", which are areas of memory that may reflect the differing physical nature of
 122831 the memory. Operations to allocate memory from these regions and pools are also
 122832 provided.

122833 Requirements

122834 Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap()*
 122835 and its related interfaces to achieve mapping and allocation of typed memory. However,
 122836 the issue of sharing typed memory (allocated or mapped) and the complication of multiple
 122837 ports are not addressed in any consistent way by existing UNIX system practice. Part of
 122838 this functionality is existing practice in specialized realtime operating systems. In order to
 122839 solidify the capabilities implied by the model above, the following requirements are
 122840 imposed on the interface:

122841 † Identification of Typed Memory Pools and Ports

122842 All processes (running in all processors) in the system are able to identify a particular
 122843 (system configured) typed memory pool accessed through a particular (system
 122844 configured) port by a name. That name is a member of a name space common to all
 122845 these processes, but need not be the same name space as that containing ordinary
 122846 pathnames. The association between memory pools/ports and corresponding names
 122847 is typically established when the system is configured. The "open" operation for
 122848 typed memory objects should be distinct from the *open()* function, for consistency
 122849 with other similar services, but implementable on top of *open()*. This implies that the
 122850 handle for a typed memory object will be a file descriptor.

122851 ‡ Allocation and Mapping of Typed Memory

122852 Once a typed memory object has been identified by a process, it is possible to both
 122853 map user-selected subareas of that object into process address space and to map
 122854 system-selected (that is, dynamically allocated) subareas of that object, with user-
 122855 specified length, into process address space. It is also possible to determine the
 122856 maximum length of memory allocation that may be requested from a given typed

122857	memory object.
122858	‡ h Assessing Typed Memory
122859	Two or more processes are able to share portions of typed memory, either user-
122860	selected or dynamically allocated. This requirement applies also to dynamically
122861	allocated regions of memory that are composed of several non-contiguous pieces.
122862	‡ o Contiguous Allocation
122863	For dynamic allocation, it is the user's option whether the system is required to
122864	allocate a contiguous subarea within the typed memory object, or whether it is
122865	permitted to allocate discontinuous fragments which appear contiguous in the
122866	process mapping. Contiguous allocation simplifies the process of sharing allocated
122867	typed memory, while discontinuous allocation allows for potentially better recovery
122868	of deallocated typed memory.
122869	‡ c Assessing Typed Memory Through Different Ports
122870	Once a subarea of a typed memory object has been mapped, it is possible to
122871	determine the location and length corresponding to a user-selected portion of that
122872	object within the memory pool. This location and length can then be used to remap
122873	that portion of memory for access from another port. If the referenced portion of
122874	typed memory was allocated discontinuously, the length thus determined may be
122875	shorter than anticipated, and the user code must adapt to the value returned.
122876	‡ e Allocation
122877	When a previously mapped subarea of typed memory is no longer mapped by any
122878	process in the system—as a result of a call or calls to <i>munmap()</i> ‡ that subarea
122879	becomes potentially reusable for dynamic allocation; actual reuse of the subarea is a
122880	function of the dynamic typed memory allocation policy.
122881	‡ r Allocated Mapping
122882	It must be possible to map user-selected subareas of a typed memory object without
122883	marking that subarea as unavailable for allocation. This option is not the default
122884	behavior, and requires appropriate privileges.
122885	Scenario
122886	The following scenario will serve to clarify the use of the typed memory interfaces.
122887	Process A running on P_1 (see Figure B-1 , on page 3605) wants to allocate some memory
122888	from memory pool $M_{2'}$, and it wants to share this portion of memory with process B
122889	running on P_2 . Since P_2 only has access to the lower part of $M_{2'}$, both processes will use the
122890	memory pool named M_{2b} , which is the part of M_2 that is accessible both from P_1 and P_2 . The
122891	operations that both processes need to perform are shown below:
122892	‡ l Allocating Typed Memory
122893	Process A calls <i>posix_typed_mem_open()</i> with the name /typed.m2b-b1 and a <i>tflag</i> of
122894	POSIX_TYPED_MEM_ALLOCATE to get a file descriptor usable for allocating from
122895	pool M_{2b} , accessed through port B_1 . It then calls <i>mmap()</i> with this file descriptor
122896	requesting a length of 4 096 bytes. The system allocates two discontinuous blocks of
122897	sizes 1 024 and 3 072 bytes within M_{2b} . The <i>mmap()</i> function returns a pointer to a
122898	4 096-byte array in process A's logical address space, mapping the allocated blocks
122899	contiguously. Process A can then utilize the array, and store data in it.

122900 ‡ Determining the Location of the Allocated Blocks

122901 Process A can determine the lengths and offsets (relative to M_{2b}) of the two blocks
 122902 allocated, by using the following procedure: First, process A calls `posix_mem_offset()`
 122903 with the address of the first element of the array and length 4096. Upon return, the
 122904 offset and length (1024 bytes) of the first block are returned. A second call to
 122905 `posix_mem_offset()` is then made using the address of the first element of the array
 122906 plus 1024 (the length of the first block), and a new length of 4096–1024. If there were
 122907 more fragments allocated, this procedure could have been continued within a loop
 122908 until the offsets and lengths of all the blocks were obtained. Notice that this relatively
 122909 complex procedure can be avoided if contiguous allocation is requested (by opening
 122910 the typed memory object with the `tflag`
 122911 `POSIX_TYPED_MEM_ALLOCATE_CONTIG`).

122912 ‡ Passing Data Across Processes

122913 Process A passes the two offset values and lengths obtained from the
 122914 `posix_mem_offset()` calls to process B running on P_2 , via some form of interprocess
 122915 communication. Process B can gain access to process A's data by calling
 122916 `posix_typed_mem_open()` with the name `/typed.m2b-b2` and a `tflag` of zero, then using
 122917 two `mmap()` calls on the resulting file descriptor to map the two subareas of that
 122918 typed memory object to its own address space.

122919 Rationale for no `mem_alloc()` and `mem_free()`

122920 The standard developers had originally proposed a pair of new flags to `mmap()` which,
 122921 when applied to a typed memory object descriptor, would cause `mmap()` to allocate
 122922 dynamically from an unallocated and unmapped area of the typed memory object.
 122923 Deallocation was similarly accomplished through the use of `munmap()`. This was rejected
 122924 by the ballot group because it excessively complicated the (already rather complex)
 122925 `mmap()` interface and introduced semantics useful only for typed memory, to a function
 122926 which must also map shared memory and files. They felt that a memory allocator should
 122927 be built on top of `mmap()` instead of being incorporated within the same interface, much as
 122928 the ISO C standard libraries build `malloc()` on top of the virtual memory mapping
 122929 functions `brk()` and `sbrk()`. This would eliminate the complicated semantics involved with
 122930 unmapping only part of an allocated block of typed memory.

122931 To attempt to achieve ballot group consensus, typed memory allocation and deallocation
 122932 was first migrated from `mmap()` and `munmap()` to a pair of complementary functions
 122933 modeled on the ISO C standard `malloc()` and `free()`. The `mem_alloc()` function specified
 122934 explicitly the typed memory object (typed memory pool/access port) from which
 122935 allocation takes place, unlike `malloc()` where the memory pool and port are unspecified.
 122936 The `mem_free()` function handled deallocation. These new semantics still met all of the
 122937 requirements detailed above without modifying the behavior of `mmap()` except to allow it
 122938 to map specified areas of typed memory objects. An implementation would have been free
 122939 to implement `mem_alloc()` and `mem_free()` over `mmap()`, through `mmap()`, or independently
 122940 but cooperating with `mmap()`.

122941 The ballot group was queried to see if this was an acceptable alternative, and while there
 122942 was some agreement that it achieved the goal of removing the complicated semantics of
 122943 allocation from the `mmap()` interface, several balloters realized that it just created two
 122944 additional functions that behaved, in great part, like `mmap()`. These balloters proposed an
 122945 alternative which has been implemented here in place of a separate `mem_alloc()` and
 122946 `mem_free()`. This alternative is based on four specific suggestions:

- 122947 1. The `posix_typed_mem_open()` function should provide a flag which specifies
 122948 ``allocate on `mmap()`'' (otherwise, `mmap()` just maps the underlying object). This
 122949 allows things roughly similar to `/dev/zero` versus `/dev/swap`. Two such flags have
 122950 been implemented, one of which forces contiguous allocation.
- 122951 2. The `posix_mem_offset()` function is acceptable because it can be applied usefully to
 122952 mapped objects in general. It should return the file descriptor of the underlying
 122953 object.
- 122954 3. The `mem_get_info()` function in an earlier draft should be renamed
 122955 `posix_typed_mem_get_info()` because it is not generally applicable to memory objects.
 122956 It should probably return the file descriptor's allocation attribute. The renaming of
 122957 the function has been implemented, but having it return a piece of information
 122958 which is readily known by an application without this function has been rejected.
 122959 Its whole purpose is to query the typed memory object for attributes that are not
 122960 user-specified, but determined by the implementation.
- 122961 4. There should be no separate `mem_alloc()` or `mem_free()` functions. Instead, using
 122962 `mmap()` on a typed memory object opened with an ``allocate on `mmap()`'' flag
 122963 should be used to force allocation. These are precisely the semantics defined in the
 122964 current draft.

122965 Rationale for no Typed Memory Access Management

122966 The working group had originally defined an additional interface (and an additional kind
 122967 of object: typed memory master) to establish and dissolve mappings to typed memory on
 122968 behalf of devices or processors which were independent of the operating system and had
 122969 no inherent capability to directly establish mappings on their own. This was to have
 122970 provided functionality similar to device driver interfaces such as `physio()` and their
 122971 underlying bus-specific interfaces (for example, `mballoc()`) which serve to set up and break
 122972 down DMA pathways, and derive mapped addresses for use by hardware devices and
 122973 processor cards.

122974 The ballot group felt that this was beyond the scope of POSIX.1 and its amendments.
 122975 Furthermore, the removal of interrupt handling interfaces from a preceding amendment
 122976 (the IEEE Std 1003.1d-1999) during its balloting process renders these typed memory
 122977 access management interfaces an incomplete solution to portable device management from
 122978 a user process; it would be possible to initiate a device transfer to/from typed memory, but
 122979 impossible to handle the transfer-complete interrupt in a portable way.

122980 To achieve ballot group consensus, all references to typed memory access management
 122981 capabilities were removed. The concept of portable interfaces from a device driver to both
 122982 operating system and hardware is being addressed by the Uniform Driver Interface (UDI)
 122983 industry forum, with formal standardization deferred until proof of concept and industry-
 122984 wide acceptance and implementation.

122985 B.2.8.4 Process Scheduling

122986 IEEE PASC Interpretation 1003.1 #96 has been applied, adding the `pthread_setschedprio()`
 122987 function. This was added since previously there was no way for a thread to lower its own
 122988 priority without going to the tail of the threads list for its new priority. This capability is
 122989 necessary to bound the duration of priority inversion encountered by a thread.

122990 The following portion of the rationale presents models, requirements, and standardization
 122991 issues relevant to process scheduling; see also [Section B.2.9.4](#) (on page 3651).

122992 In an operating system supporting multiple concurrent processes, the system determines the

122993 order in which processes execute to meet implementation-defined goals. For time-sharing
122994 systems, the goal is to enhance system throughput and promote fairness; the application is
122995 provided with little or no control over this sequencing function. While this is acceptable and
122996 desirable behavior in a time-sharing system, it is inappropriate in a realtime system; realtime
122997 applications must specifically control the execution sequence of their concurrent processes in
122998 order to meet externally defined response requirements.

122999 In POSIX.1-2017, the control over process sequencing is provided using a concept of scheduling
123000 policies. These policies, described in detail in this section, define the behavior of the system
123001 whenever processor resources are to be allocated to competing processes. Only the behavior of
123002 the policy is defined; conforming implementations are free to use any mechanism desired to
123003 achieve the described behavior.

123004 Models

123005 In an operating system supporting multiple concurrent processes, the system determines
123006 the order in which processes execute and might force long-running processes to yield to
123007 other processes at certain intervals. Typically, the scheduling code is executed whenever an
123008 event occurs that might alter the process to be executed next.

123009 The simplest scheduling strategy is a “first-in, first-out” (FIFO) dispatcher. Whenever a
123010 process becomes runnable, it is placed on the end of a ready list. The process at the front of
123011 the ready list is executed until it exits or becomes blocked, at which point it is removed
123012 from the list. This scheduling technique is also known as “run-to-completion” or “run-to-
123013 block”.

123014 A natural extension to this scheduling technique is the assignment of a “non-migrating
123015 priority” to each process. This policy differs from strict FIFO scheduling in only one
123016 respect: whenever a process becomes runnable, it is placed at the end of the list of
123017 processes runnable at that priority level. When selecting a process to run, the system
123018 always selects the first process from the highest priority queue with a runnable process.
123019 Thus, when a process becomes unblocked, it will preempt a running process of lower
123020 priority without otherwise altering the ready list. Further, if a process elects to alter its
123021 priority, it is removed from the ready list and reinserted, using its new priority, according
123022 to the policy above.

123023 While the above policy might be considered unfriendly in a time-sharing environment in
123024 which multiple users require more balanced resource allocation, it could be ideal in a
123025 realtime environment for several reasons. The most important of these is that it is
123026 deterministic: the highest-priority process is always run and, among processes of equal
123027 priority, the process that has been runnable for the longest time is executed first. Because
123028 of this determinism, cooperating processes can implement more complex scheduling simply
123029 by altering their priority. For instance, if processes at a single priority were to reschedule
123030 themselves at fixed time intervals, a time-slice policy would result.

123031 In a dedicated operating system in which all processes are well-behaved realtime
123032 applications, non-migrating priority scheduling is sufficient. However, many existing
123033 implementations provide for more complex scheduling policies.

123034 POSIX.1-2017 specifies a linear scheduling model. In this model, every process in the
123035 system has a priority. The system scheduler always dispatches a process that has the
123036 highest (generally the most time-critical) priority among all runnable processes in the
123037 system. As long as there is only one such process, the dispatching policy is trivial. When
123038 multiple processes of equal priority are eligible to run, they are ordered according to a
123039 strict run-to-completion (FIFO) policy.

123040 The priority is represented as a positive integer and is inherited from the parent process.

123041 For processes running under a fixed priority scheduling policy, the priority is never altered
123042 except by an explicit function call.

123043 It was determined arbitrarily that larger integers correspond to “higher priorities”.

123044 Certain implementations might impose restrictions on the priority ranges to which
123045 processes can be assigned. There also can be restrictions on the set of policies to which
123046 processes can be set.

123047 Requirements

123048 Realtime processes require that scheduling be fast and deterministic, and that it guarantees
123049 to preempt lower priority processes.

123050 Thus, given the linear scheduling model, realtime processes require that they be run at a
123051 priority that is higher than other processes. Within this framework, realtime processes are
123052 free to yield execution resources to each other in a completely portable and
123053 implementation-defined manner.

123054 As there is a generally perceived requirement for processes at the same priority level to
123055 share processor resources more equitably, provisions are made by providing a scheduling
123056 policy (that is, SCHED_RR) intended to provide a timeslice-like facility.

123057 **Note:** The following topics assume that low numeric priority implies low scheduling criticality
123058 and *vice versa*.

123059 Rationale for New Interface

123060 Realtime applications need to be able to determine when processes will run in relation to
123061 each other. It must be possible to guarantee that a critical process will run whenever it is
123062 runnable; that is, whenever it wants to for as long as it needs. SCHED_FIFO satisfies this
123063 requirement. Additionally, SCHED_RR was defined to meet a realtime requirement for a
123064 well-defined time-sharing policy for processes at the same priority.

123065 It would be possible to use the BSD *setpriority()* and *getpriority()* functions by redefining
123066 the meaning of the “nice” parameter according to the scheduling policy currently in use by
123067 the process. The System V *nice()* interface was felt to be undesirable for realtime because it
123068 specifies an adjustment to the “nice” value, rather than setting it to an explicit value.
123069 Realtime applications will usually want to set priority to an explicit value. Also, System V
123070 *nice()* does not allow for changing the priority of another process.

123071 With the POSIX.1b interfaces, the traditional “nice” value does not affect the SCHED_FIFO
123072 or SCHED_RR scheduling policies. If a “nice” value is supported, it is implementation-
123073 defined whether it affects the SCHED_OTHER policy.

123074 An important aspect of POSIX.1-2017 is the explicit description of the queuing and
123075 preemption rules. It is critical, to achieve deterministic scheduling, that such rules be
123076 stated clearly in POSIX.1-2017.

123077 POSIX.1-2017 does not address the interaction between priority and swapping. The issues
123078 involved with swapping and virtual memory paging are extremely implementation-
123079 defined and would be nearly impossible to standardize at this point. The proposed
123080 scheduling paradigm, however, fully describes the scheduling behavior of runnable
123081 processes, of which one criterion is that the working set be resident in memory. Assuming
123082 the existence of a portable interface for locking portions of a process in memory, paging
123083 behavior need not affect the scheduling of realtime processes.

123084 POSIX.1-2017 also does not address the priorities of “system” processes. In general, these
123085 processes should always execute in low-priority ranges to avoid conflict with other
123086 realtime processes. Implementations should document the priority ranges in which system

123087 processes run.

123088 The default scheduling policy is not defined. The effect of I/O interrupts and other system
123089 processing activities is not defined. The temporary lending of priority from one process to
123090 another (such as for the purposes of affecting freeing resources) by the system is not
123091 addressed. Preemption of resources is not addressed. Restrictions on the ability of a
123092 process to affect other processes beyond a certain level (influence levels) is not addressed.

123093 The rationale used to justify the simple time-quantum scheduler is that it is common
123094 practice to depend upon this type of scheduling to ensure “fair” distribution of processor
123095 resources among portions of the application that must interoperate in a serial fashion. Note
123096 that POSIX.1-2017 is silent with respect to the setting of this time quantum, or whether it is
123097 a system-wide value or a per-process value, although it appears that the prevailing
123098 realtime practice is for it to be a system-wide value.

123099 In a system with N processes at a given priority, all processor-bound, in which the time
123100 quantum is equal for all processes at a specific priority level, the following assumptions
123101 are made of such a scheduling policy:

- 123102 1. A time quantum Q exists and the current process will own control of the processor
123103 for at least a duration of Q and will have the processor for a duration of Q .
- 123104 2. The N th process at that priority will control a processor within a duration of $(N-1)$
123105 $\times Q$.

123106 These assumptions are necessary to provide equal access to the processor and bounded
123107 response from the application.

123108 The assumptions hold for the described scheduling policy only if no system overhead,
123109 such as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one
123110 of the two assumptions becomes fallacious, based upon how Q is measured by the system.

123111 If Q is measured by clock time, then the assumption that the process obtains a duration Q
123112 processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed
123113 with N processes in which a single process undergoes complete processor starvation if a
123114 peripheral device, such as an analog-to-digital converter, generates significant interrupt
123115 activity periodically with a period of $N \times Q$.

123116 If Q is measured as actual processor time, then the assumption that the N th process runs in
123117 within the duration $(N-1) \times Q$ is false.

123118 It should be noted that SCHED_FIFO suffers from interrupt-based delay as well. However,
123119 for SCHED_FIFO, the implied response of the system is “as soon as possible”, so that the
123120 interrupt load for this case is a vendor selection and not a compliance issue.

123121 With this in mind, it is necessary either to complete the definition by including bounds on
123122 the interrupt load, or to modify the assumptions that can be made about the scheduling
123123 policy.

123124 Since the motivation of inclusion of the policy is common usage, and since current
123125 applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient
123126 to express existing application needs and is less restrictive in the standard definition. No
123127 difference in interface is necessary.

123128 In an implementation in which the time quantum is equal for all processes at a specific
123129 priority, our assumptions can then be restated as:

123130 ‡ A time quantum Q exists, and a processor-bound process will be rescheduled after a
123131 duration of, at most, Q . Time quantum Q may be defined in either wall clock time or
123132 execution time.

123133 In general, the N th process of a priority level should wait no longer than $(N-1) \times Q$
 123134 time to execute, assuming no processes exist at higher priority levels.

123135 ‡ A process should wait indefinitely.

123136 For implementations supporting per-process time quanta, these assumptions can be
 123137 readily extended.

123138 Sporadic Server Scheduling Policy

123139 The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical
 123140 realtime systems. This mechanism reserves a certain bounded amount of execution capacity for
 123141 processing aperiodic events at a high priority level. Any aperiodic events that cannot be
 123142 processed within the bounded amount of execution capacity are executed in the background at a
 123143 low priority level. Thus, a certain amount of execution capacity can be guaranteed to be
 123144 available for processing periodic tasks, even under burst conditions in the arrival of aperiodic
 123145 processing requests (that is, a large number of requests in a short time interval). The sporadic
 123146 server also simplifies the schedulability analysis of the realtime system, because it allows
 123147 aperiodic processes or threads to be treated as if they were periodic. The sporadic server was
 123148 first described by Sprunt, et al.

123149 The key concept of the sporadic server is to provide and limit a certain amount of computation
 123150 capacity for processing aperiodic events at their assigned normal priority, during a time interval
 123151 called the “replenishment period”. Once the entity controlled by the sporadic server mechanism
 123152 is initialized with its period and execution-time budget attributes, it preserves its execution
 123153 capacity until an aperiodic request arrives. The request will be serviced (if there are no higher
 123154 priority activities pending) as long as there is execution capacity left. If the request is completed,
 123155 the actual execution time used to service it is subtracted from the capacity, and a replenishment
 123156 of this amount of execution time is scheduled to happen one replenishment period after the
 123157 arrival of the aperiodic request. If the request is not completed, because there is no execution
 123158 capacity left, then the aperiodic process or thread is assigned a lower background priority. For
 123159 each portion of consumed execution capacity the execution time used is replenished after one
 123160 replenishment period. At the time of replenishment, if the sporadic server was executing at a
 123161 background priority level, its priority is elevated to the normal level. Other similar
 123162 replenishment policies have been defined, but the one presented here represents a compromise
 123163 between efficiency and implementation complexity.

123164 The interface that appears in this section defines a new scheduling policy for threads and
 123165 processes that behaves according to the rules of the sporadic server mechanism. Scheduling
 123166 attributes are defined and functions are provided to allow the user to set and get the parameters
 123167 that control the scheduling behavior of this mechanism, namely the normal and low priority, the
 123168 replenishment period, the maximum number of pending replenishment operations, and the
 123169 initial execution-time budget.

123170 Scheduling Aperiodic Activities

123171 Virtually all realtime applications are required to process aperiodic activities. In many
 123172 cases, there are tight timing constraints that the response to the aperiodic events must
 123173 meet. Usual timing requirements imposed on the response to these events are:

123174 ‡ The effects of an aperiodic activity on the response time of lower priority activities
 123175 must be controllable and predictable.

123176 ‡ The system must provide the fastest possible response time to aperiodic events.

123177 ‡ It must be possible to take advantage of all the available processing bandwidth not
 123178 needed by time-critical activities to enhance average-case response times to aperiodic
 123179 events.

- 123180 Traditional methods for scheduling aperiodic activities are background processing, polling
123181 tasks, and direct event execution:
- 123182 ‡ aC Background processing consists of assigning a very low priority to the processing of
123183 aperiodic events. It utilizes all the available bandwidth in the system that has not
123184 been consumed by higher priority threads. However, it is very difficult, or
123185 impossible, to meet requirements on average-case response time, because the
123186 aperiodic entity has to wait for the execution of all other entities which have higher
123187 priority.
- 123188 ‡ oL Polling consists of creating a periodic process or thread for servicing aperiodic
123189 requests. At regular intervals, the polling entity is started and its services
123190 accumulated pending aperiodic requests. If no aperiodic requests are pending, the
123191 polling entity suspends itself until its next period. Polling allows the aperiodic
123192 requests to be processed at a higher priority level. However, worst and average-case
123193 response times of polling entities are a direct function of the polling period, and there
123194 is execution overhead for each polling period, even if no event has arrived. If the
123195 deadline of the aperiodic activity is short compared to the inter-arrival time, the
123196 polling frequency must be increased to guarantee meeting the deadline. For this case,
123197 the increase in frequency can dramatically reduce the efficiency of the system and,
123198 therefore, its capacity to meet all deadlines. Yet, polling represents a good way to
123199 handle a large class of practical problems because it preserves system predictability,
123200 and because the amortized overhead drops as load increases.
- 123201 ‡ iE Direct event execution consists of executing the aperiodic events at a high fixed-
123202 priority level. Typically, the aperiodic event is processed by an interrupt service
123203 routine as soon as it arrives. This technique provides predictable response times for
123204 aperiodic events, but makes the response times of all lower priority activities
123205 completely unpredictable under burst arrival conditions. Therefore, if the density of
123206 aperiodic event arrivals is unbounded, it may be a dangerous technique for time-
123207 critical systems. Yet, for those cases in which the physics of the system imposes a
123208 bound on the event arrival rate, it is probably the most efficient technique.
- 123209 ‡ kE Sporadic server scheduling algorithm combines the predictability of the polling
123210 approach with the short response times of the direct event execution. Thus, it allows
123211 systems to meet an important class of application requirements that cannot be met by
123212 using the traditional approaches. Multiple sporadic servers with different attributes
123213 can be applied to the scheduling of multiple classes of aperiodic events, each with
123214 different kinds of timing requirements, such as individual deadlines, average
123215 response times, and so on. It also has many other interesting applications for
123216 realtime, such as scheduling producer/consumer tasks in time-critical systems,
123217 limiting the effects of faults on the estimation of task execution-time requirements,
123218 and so on.
- 123219 Existing Practice
- 123220 The sporadic server has been used in different kinds of applications, including military
123221 avionics, robot control systems, industrial automation systems, and so on. There are
123222 examples of many systems that cannot be successfully scheduled using the classic
123223 approaches, such as direct event execution, or polling, and are schedulable using a
123224 sporadic server scheduler. The sporadic server algorithm itself can successfully schedule
123225 all systems scheduled with direct event execution or polling.
- 123226 The sporadic server scheduling policy has been implemented as a commercial product in
123227 the run-time system of the Verdix Ada compiler. There are also many applications that
123228 have used a much less efficient application-level sporadic server. These realtime

123229 applications would benefit from a sporadic server scheduler implemented at the scheduler
123230 level.

123231 Library-Level *versus* Kernel-Level Implementation

123232 The sporadic server interface described in this section requires the sporadic server policy
123233 to be implemented at the same level as the scheduler. This means that the process sporadic
123234 server must be implemented at the kernel level and the thread sporadic server policy
123235 implemented at the same level as the thread scheduler; that is, kernel or library level.

123236 In an earlier interface for the sporadic server, this mechanism was implementable at a
123237 different level than the scheduler. This feature allowed the implementor to choose between
123238 an efficient scheduler-level implementation, or a simpler user or library-level
123239 implementation. However, the working group considered that this interface made the use
123240 of sporadic servers more complex, and that library-level implementations would lack some
123241 of the important functionality of the sporadic server, namely the limitation of the actual
123242 execution time of aperiodic activities. The working group also felt that the interface
123243 described in this chapter does not preclude library-level implementations of threads
123244 intended to provide efficient low-overhead scheduling for those threads that are not
123245 scheduled under the sporadic server policy.

123246 Range of Scheduling Priorities

123247 Each of the scheduling policies supported in POSIX.1-2017 has an associated range of
123248 priorities. The priority ranges for each policy might or might not overlap with the priority
123249 ranges of other policies. For time-critical realtime applications it is usual for periodic and
123250 aperiodic activities to be scheduled together in the same processor. Periodic activities will
123251 usually be scheduled using the SCHED_FIFO scheduling policy, while aperiodic activities
123252 may be scheduled using SCHED_SPORADIC. Since the application developer will require
123253 complete control over the relative priorities of these activities in order to meet his timing
123254 requirements, it would be desirable for the priority ranges of SCHED_FIFO and
123255 SCHED_SPORADIC to overlap completely. Therefore, although POSIX.1-2017 does not
123256 require any particular relationship between the different priority ranges, it is
123257 recommended that these two ranges should coincide.

123258 Dynamically Setting the Sporadic Server Policy

123259 Several members of the working group requested that implementations should not be
123260 required to support dynamically setting the sporadic server scheduling policy for a thread.
123261 The reason is that this policy may have a high overhead for library-level implementations
123262 of threads, and if threads are allowed to dynamically set this policy, this overhead can be
123263 experienced even if the thread does not use that policy. By disallowing the dynamic setting
123264 of the sporadic server scheduling policy, these implementations can accomplish efficient
123265 scheduling for threads using other policies. If a strictly conforming application needs to
123266 use the sporadic server policy, and is therefore willing to pay the overhead, it must set this
123267 policy at the time of thread creation.

123268 Limitation of the Number of Pending Replenishments

123269 The number of simultaneously pending replenishment operations must be limited for each
123270 sporadic server for two reasons: an unlimited number of replenishment operations would
123271 need an unlimited number of system resources to store all the pending replenishment
123272 operations; on the other hand, in some implementations each replenishment operation will
123273 represent a source of priority inversion (just for the duration of the replenishment
123274 operation) and thus, the maximum amount of replenishments must be bounded to
123275 guarantee bounded response times. The way in which the number of replenishments is
123276 bounded is by lowering the priority of the sporadic server to *sched_ss_low_priority* when

123277 the number of pending replenishments has reached its limit. In this way, no new
123278 replenishments are scheduled until the number of pending replenishments decreases.

123279 In the sporadic server scheduling policy defined in POSIX.1-2017, the application can
123280 specify the maximum number of pending replenishment operations for a single sporadic
123281 server, by setting the value of the *sched_ss_max_repl* scheduling parameter. This value must
123282 be between one and {SS_REPL_MAX}, which is a maximum limit imposed by the
123283 implementation. The limit {SS_REPL_MAX} must be greater than or equal to
123284 {_POSIX_SS_REPL_MAX}, which is defined to be four in POSIX.1-2017. The minimum
123285 limit of four was chosen so that an application can at least guarantee that four different
123286 aperiodic events can be processed during each interval of length equal to the
123287 replenishment period.

123288 B.2.8.5 Clocks and Timers

123289 Clocks

123290 POSIX.1-2017 and the ISO C standard both define functions for obtaining system time.
123291 Implicit behind these functions is a mechanism for measuring passage of time. This
123292 specification makes this mechanism explicit and calls it a clock. The CLOCK_REALTIME
123293 clock required by POSIX.1-2017 is a higher resolution version of the clock that maintains
123294 POSIX.1 system time. This is a “system-wide” clock, in that it is visible to all processes
123295 and, were it possible for multiple processes to all read the clock at the same time, they
123296 would see the same value.

123297 An extensible interface was defined, with the ability for implementations to define
123298 additional clocks. This was done because of the observation that many realtime platforms
123299 support multiple clocks, and it was desired to fit this model within the standard interface.
123300 But implementation-defined clocks need not represent actual hardware devices, nor are
123301 they necessarily system-wide.

123302 Timers

123303 Two timer types are required for a system to support realtime applications:

123304 1. One-shot

123305 A one-shot timer is a timer that is armed with an initial expiration time, either
123306 relative to the current time or at an absolute time (based on some timing base, such
123307 as time in seconds and nanoseconds since the Epoch). The timer expires once and
123308 then is disarmed. With the specified facilities, this is accomplished by setting the
123309 *it_value* member of the *value* argument to the desired expiration time and the
123310 *it_interval* member to zero.

123311 2. Periodic

123312 A periodic timer is a timer that is armed with an initial expiration time, again either
123313 relative or absolute, and a repetition interval. When the initial expiration occurs,
123314 the timer is reloaded with the repetition interval and continues counting. With the
123315 specified facilities, this is accomplished by setting the *it_value* member of the *value*
123316 argument to the desired initial expiration time and the *it_interval* member to the
123317 desired repetition interval.

123318 For both of these types of timers, the time of the initial timer expiration can be specified in
123319 two ways:

- 123320 1. Relative (to the current time)
- 123321 2. Absolute

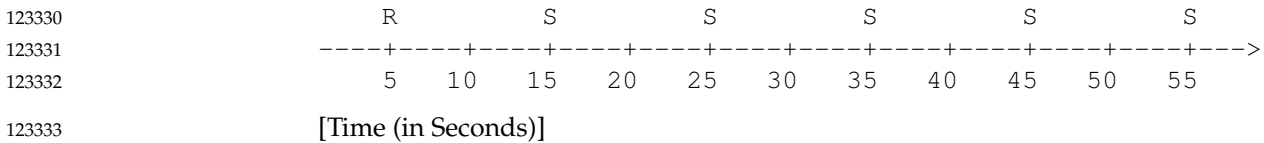
123322 Examples of Using Realtime Timers

123323 In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method
 123324 request, and *E* suggests an internal operating system event.

123325 ‡ Periodic Timer: Data Logging

123326 During an experiment, it might be necessary to log realtime data periodically to an
 123327 internal buffer or to a mass storage device. With a periodic scheduling method, a
 123328 logging module can be started automatically at fixed time intervals to log the data.

123329 Program schedule is requested every 10 seconds.

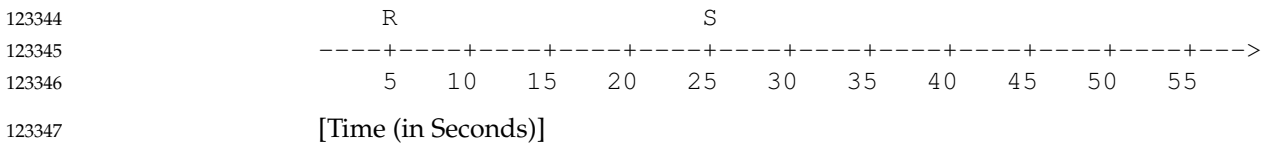


123333 To achieve this type of scheduling using the specified facilities, one would allocate a
 123334 per-process timer based on clock ID CLOCK_REALTIME. Then the timer would be
 123335 armed via a call to *timer_settime()* with the TIMER_ABSTIME flag reset, and with an
 123336 initial expiration value and a repetition interval of 10 seconds.
 123337

123338 ‡ One-shot Timer (Relative Time): Device Initialization

123339 In an emission test environment, large sample bags are used to capture the exhaust
 123340 from a vehicle. The exhaust is purged from these bags before each and every test.
 123341 With a one-shot timer, a module could initiate the purge function and then suspend
 123342 itself for a predetermined period of time while the sample bags are prepared.

123343 Program schedule requested 20 seconds after call is issued.



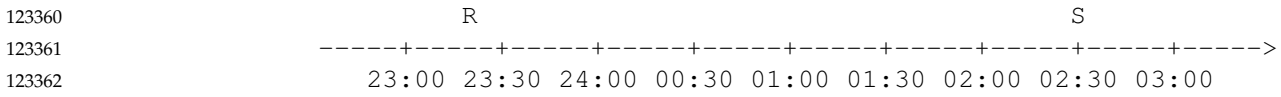
123347 To achieve this type of scheduling using the specified facilities, one would allocate a
 123348 per-process timer based on clock ID CLOCK_REALTIME. Then the timer would be
 123349 armed via a call to *timer_settime()* with the TIMER_ABSTIME flag reset, and with an
 123350 initial expiration value of 20 seconds and a repetition interval of zero.
 123351

123352 Note that if the program wishes merely to suspend itself for the specified interval, it
 123353 could more easily use *nanosleep()*.

123354 ‡ One-shot Timer (Absolute Time): Data Transmission

123355 The results from an experiment are often moved to a different system within a
 123356 network for post-processing or archiving. With an absolute one-shot timer, a module
 123357 that moves data from a test-cell computer to a host computer can be automatically
 123358 scheduled on a daily basis.

123359 Program schedule requested for 2:30 a.m.



123408 process virtual time clock—timer expirations could be requested when the process has
 123409 used a specified total amount of virtual time (absolute), or when it has used a specified
 123410 *additional* amount of virtual time (relative).

123411 The interfaces also allow flexibility in the implementation of the functions. For example, an
 123412 implementation could convert all absolute times to intervals by subtracting the clock value
 123413 at the time of the call from the requested expiration time and “counting down” at the
 123414 supported resolution. Or it could convert all relative times to absolute expiration time by
 123415 adding in the clock value at the time of the call and comparing the clock value to the
 123416 expiration time at the supported resolution. Or it might even choose to maintain absolute
 123417 times as absolute and compare them to the clock value at the supported resolution for
 123418 absolute timers, and maintain relative times as intervals and count them down at the
 123419 resolution supported for relative timers. The choice will be driven by efficiency
 123420 considerations and the underlying hardware or software clock implementation.

123421 Data Definitions for Clocks and Timers

123422 POSIX.1-2017 uses a time representation capable of supporting nanosecond resolution
 123423 timers for the following reasons:

123424 † Enable POSIX.1-2017 to represent those computer systems already using
 123425 nanosecond or submicrosecond resolution clocks.

123426 † Accommodate those per-process timers that might need nanoseconds to specify an
 123427 absolute value of system-wide clocks, even though the resolution of the per-process
 123428 timer may only be milliseconds, or *vice versa*.

123429 † Use the number of nanoseconds in a second can be represented in 32 bits.

123430 Time values are represented in the **timespec** structure. The *tv_sec* member is of type **time_t**
 123431 so that this member is compatible with time values used by POSIX.1 functions and the
 123432 ISO C standard. The *tv_nsec* member is a **signed long** in order to simplify and clarify code
 123433 that decrements or finds differences of time values. Note that because 1 billion (number of
 123434 nanoseconds per second) is less than half of the value representable by a signed 32-bit
 123435 value, it is always possible to add two valid fractional seconds represented as integral
 123436 nanoseconds without overflowing the signed 32-bit value.

123437 A maximum allowable resolution for the CLOCK_REALTIME clock of 20 ms (1/50
 123438 seconds) was chosen to allow line frequency clocks in European countries to be
 123439 conforming. 60 Hz clocks in the US will also be conforming, as will finer granularity
 123440 clocks, although a Strictly Conforming Application cannot assume a granularity of less
 123441 than 20 ms (1/50 seconds).

123442 The minimum allowable maximum time allowed for the CLOCK_REALTIME clock and
 123443 the function *nanosleep()*, and timers created with *clock_id=CLOCK_REALTIME*, is
 123444 determined by the fact that the *tv_sec* member is of type **time_t**.

123445 POSIX.1-2017 specifies that timer expirations must not be delivered early, and *nanosleep()*
 123446 must not return early due to quantization error. POSIX.1-2017 discusses the various
 123447 implementations of *alarm()* in the rationale and states that implementations that do not
 123448 allow alarm signals to occur early are the most appropriate, but refrained from mandating
 123449 this behavior. Because of the importance of predictability to realtime applications,
 123450 POSIX.1-2017 takes a stronger stance.

123451 The standard developers considered using a time representation that differs from
 123452 POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field as a
 123453 fractional second in nanoseconds, the other methodology defines this as a binary fraction
 123454 of one second, with the radix point assumed before the most significant bit.

123455 POSIX.1b is a software, source-level standard and most of the benefits of the alternate
123456 representation are enjoyed by hardware implementations of clocks and algorithms. It was
123457 felt that mandating this format for POSIX.1b clocks and timers would unnecessarily
123458 burden the application developer with writing, possibly non-portable, multiple precision
123459 arithmetic packages to perform conversion between binary fractions and integral units
123460 such as nanoseconds, milliseconds, and so on.

123461 **Rationale for the Monotonic Clock**

123462 For those applications that use time services to achieve realtime behavior, changing the value of
123463 the clock on which these services rely may cause erroneous timing behavior. For these
123464 applications, it is necessary to have a monotonic clock which cannot run backwards, and which
123465 has a maximum clock jump that is required to be documented by the implementation.
123466 Additionally, it is desirable (but not required by POSIX.1-2017) that the monotonic clock
123467 increases its value uniformly. This clock should not be affected by changes to the system time;
123468 for example, to synchronize the clock with an external source or to account for leap seconds.
123469 Such changes would cause errors in the measurement of time intervals for those time services
123470 that use the absolute value of the clock.

123471 One could argue that by defining the behavior of time services when the value of a clock is
123472 changed, deterministic realtime behavior can be achieved. For example, one could specify that
123473 relative time services should be unaffected by changes in the value of a clock. However, there are
123474 time services that are based upon an absolute time, but that are essentially intended as relative
123475 time services. For example, *pthread_cond_timedwait()* uses an absolute time to allow it to wake
123476 up after the required interval despite spurious wakeups. Although sometimes the
123477 *pthread_cond_timedwait()* timeouts are absolute in nature, there are many occasions in which they
123478 are relative, and their absolute value is determined from the current time plus a relative time
123479 interval. In this latter case, if the clock changes while the thread is waiting, the wait interval will
123480 not be the expected length. If a *pthread_cond_timedwait()* function were created that would take a
123481 relative time, it would not solve the problem because to retain the intended “deadline” a thread
123482 would need to compensate for latency due to the spurious wakeup, and preemption between
123483 wakeup and the next wait.

123484 The solution is to create a new monotonic clock, whose value does not change except for the
123485 regular ticking of the clock, and use this clock for implementing the various relative timeouts
123486 that appear in the different POSIX interfaces, as well as allow *pthread_cond_timedwait()* to choose
123487 this new clock for its timeout. A new *clock_nanosleep()* function is created to allow an application
123488 to take advantage of this newly defined clock. Notice that the monotonic clock may be
123489 implemented using the same hardware clock as the system clock.

123490 Relative timeouts for *sigtimedwait()* and *aio_suspend()* have been redefined to use the monotonic
123491 clock, if present. The *alarm()* function has not been redefined, because the same effect but with
123492 better resolution can be achieved by creating a timer (for which the appropriate clock may be
123493 chosen).

123494 The *pthread_cond_timedwait()* function has been treated in a different way, compared to other
123495 functions with absolute timeouts, because it is used to wait for an event, and thus it may have a
123496 deadline, while the other timeouts are generally used as an error recovery mechanism, and for
123497 them the use of the monotonic clock is not so important. Since the desired timeout for the
123498 *pthread_cond_timedwait()* function may either be a relative interval or an absolute time of day
123499 deadline, a new initialization attribute has been created for condition variables to specify the
123500 clock that is used for measuring the timeout in a call to *pthread_cond_timedwait()*. In this way, if
123501 a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is
123502 required instead, the *CLOCK_REALTIME* or another appropriate clock may be used. This
123503 capability has not been added to other functions with absolute timeouts because for those

123504 functions the expected use of the timeout is mostly to prevent errors, and not so often to meet
 123505 precise deadlines. As a consequence, the complexity of adding this capability is not justified by
 123506 its perceived application usage.

123507 The *nanosleep()* function has not been modified with the introduction of the monotonic clock.
 123508 Instead, a new *clock_nanosleep()* function has been created, in which the desired clock may be
 123509 specified in the function call.

123510 History of Resolution Issues

123511 Due to the shift from relative to absolute timeouts in IEEE Std 1003.1d-1999, the
 123512 amendments to the *sem_timedwait()*, *pthread_mutex_timedlock()*, *mq_timedreceive()*, and
 123513 *mq_timedsend()* functions of that standard have been removed. Those amendments
 123514 specified that CLOCK_MONOTONIC would be used for the (relative) timeouts if the
 123515 Monotonic Clock option was supported.

123516 Having these functions continue to be tied solely to CLOCK_MONOTONIC would not
 123517 work. Since the absolute value of a time value obtained from CLOCK_MONOTONIC is
 123518 unspecified, under the absolute timeouts interface, applications would behave differently
 123519 depending on whether the Monotonic Clock option was supported or not (because the
 123520 absolute value of the clock would have different meanings in either case).

123521 Two options were considered:

- 123522 1. Leave the current behavior unchanged, which specifies the CLOCK_REALTIME
 123523 clock for these (absolute) timeouts, to allow portability of applications between
 123524 implementations supporting or not the Monotonic Clock option.
- 123525 2. Modify these functions in the way that *pthread_cond_timedwait()* was modified to
 123526 allow a choice of clock, so that an application could use CLOCK_REALTIME when
 123527 it is trying to achieve an absolute timeout and CLOCK_MONOTONIC when it is
 123528 trying to achieve a relative timeout.

123529 It was decided that the features of CLOCK_MONOTONIC are not as critical to these
 123530 functions as they are to *pthread_cond_timedwait()*. The *pthread_cond_timedwait()* function is
 123531 given a relative timeout; the timeout may represent a deadline for an event. When these
 123532 functions are given relative timeouts, the timeouts are typically for error recovery
 123533 purposes and need not be so precise.

123534 Therefore, it was decided that these functions should be tied to CLOCK_REALTIME and
 123535 not complicated by being given a choice of clock.

123536 Execution Time Monitoring

123537 Introduction

123538 The main goals of the execution time monitoring facilities defined in this chapter are to
 123539 measure the execution time of processes and threads and to allow an application to
 123540 establish CPU time limits for these entities.

123541 The analysis phase of time-critical realtime systems often relies on the measurement of
 123542 execution times of individual threads or processes to determine whether the timing
 123543 requirements will be met. Also, performance analysis techniques for soft deadline realtime
 123544 systems rely heavily on the determination of these execution times. The execution time
 123545 monitoring functions provide application developers with the ability to measure these
 123546 execution times online and open the possibility of dynamic execution-time analysis and
 123547 system reconfiguration, if required.

123548 The second goal of allowing an application to establish execution time limits for individual

123549 processes or threads and detecting when they overrun allows program robustness to be
123550 increased by enabling online checking of the execution times.

123551 If errors are detected—possibly because of erroneous program constructs, the existence of
123552 errors in the analysis phase, or a burst of event arrivals—online detection and recovery is
123553 possible in a portable way. This feature can be extremely important for many time-critical
123554 applications. Other applications require trapping CPU-time errors as a normal way to exit
123555 an algorithm; for instance, some realtime artificial intelligence applications trigger a
123556 number of independent inference processes of varying accuracy and speed, limit how long
123557 they can run, and pick the best answer available when time runs out. In many periodic
123558 systems, overrun processes are simply restarted in the next resource period, after necessary
123559 end-of-period actions have been taken. This allows algorithms that are inherently data-
123560 dependent to be made predictable.

123561 The interface that appears in this chapter defines a new type of clock, the CPU-time clock,
123562 which measures execution time. Each process or thread can invoke the clock and timer
123563 functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-
123564 time clock of other processes or threads to enable remote monitoring of these clocks.
123565 Monitoring of threads of other processes is not supported, since these threads are not
123566 visible from outside of their own process with the interfaces defined in POSIX.1.

123567 Execution Time Monitoring Interface

123568 The clock and timer interface defined in POSIX.1 historically only defined one clock, which
123569 measures wall-clock time. The requirements for measuring execution time of processes and
123570 threads, and setting limits to their execution time by detecting when they overrun, can be
123571 accomplished with that interface if a new kind of clock is defined. These new clocks
123572 measure execution time, and one is associated with each process and with each thread. The
123573 clock functions currently defined in POSIX.1 can be used to read and set these CPU-time
123574 clocks, and timers can be created using these clocks as their timing base. These timers can
123575 then be used to send a signal when some specified execution time has been exceeded. The
123576 CPU-time clocks of each process or thread can be accessed by using the symbols
123577 `CLOCK_PROCESS_CPUTIME_ID` or `CLOCK_THREAD_CPUTIME_ID`.

123578 The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-
123579 time clock would only allow processes or threads to access their own CPU-time clocks.
123580 However, many realtime systems require the possibility of monitoring the execution time
123581 of processes or threads from independent monitoring entities. In order to allow
123582 applications to construct independent monitoring entities that do not require cooperation
123583 from or modification of the monitored entities, two functions have been added:
123584 `clock_getcpuclockid()`, for accessing CPU-time clocks of other processes, and
123585 `pthread_getcpuclockid()`, for accessing CPU-time clocks of other threads. These functions
123586 return the clock identifier associated with the process or thread specified in the call. These
123587 clock IDs can then be used in the rest of the clock function calls.

123588 The clocks accessed through these functions could also be used as a timing base for the
123589 creation of timers, thereby allowing independent monitoring entities to limit the CPU time
123590 consumed by other entities. However, this possibility would imply additional complexity
123591 and overhead because of the need to maintain a timer queue for each process or thread, to
123592 store the different expiration times associated with timers created by different processes or
123593 threads. The working group decided this additional overhead was not justified by
123594 application requirements. Therefore, creation of timers attached to the CPU-time clocks of
123595 other processes or threads has been specified as implementation-defined.

123596 Overhead Considerations

123597 The measurement of execution time may introduce additional overhead in the thread
123598 scheduling, because of the need to keep track of the time consumed by each of these
123599 entities. In library-level implementations of threads, the efficiency of scheduling could be
123600 somehow compromised because of the need to make a kernel call, at each context switch,
123601 to read the process CPU-time clock. Consequently, a thread creation attribute called *cpu-*
123602 *clock-requirement* was defined, to allow threads to disconnect their respective CPU-time
123603 clocks. However, the Ballot Group considered that this attribute itself introduced some
123604 overhead, and that in current implementations it was not worth the effort. Therefore, the
123605 attribute was deleted, and thus thread CPU-time clocks are required for all threads if the
123606 Thread CPU-Time Clocks option is supported.

123607 Accuracy of CPU-Time Clocks

123608 The mechanism used to measure the execution time of processes and threads is specified in
123609 POSIX.1-2017 as implementation-defined. The reason for this is that both the underlying
123610 hardware and the implementation architecture have a very strong influence on the
123611 accuracy achievable for measuring CPU time. For some implementations, the specification
123612 of strict accuracy requirements would represent very large overheads, or even the
123613 impossibility of being implemented.

123614 Since the mechanism for measuring execution time is implementation-defined, realtime
123615 applications will be able to take advantage of accurate implementations using a portable
123616 interface. Of course, strictly conforming applications cannot rely on any particular degree
123617 of accuracy, in the same way as they cannot rely on a very accurate measurement of wall
123618 clock time. There will always exist applications whose accuracy or efficiency requirements
123619 on the implementation are more rigid than the values defined in POSIX.1-2017 or any
123620 other standard.

123621 In any case, there is a minimum set of characteristics that realtime applications would
123622 expect from most implementations. One such characteristic is that the sum of all the
123623 execution times of all the threads in a process equals the process execution time, when no
123624 CPU-time clocks are disabled. This need not always be the case because implementations
123625 may differ in how they account for time during context switches. Another characteristic is
123626 that the sum of the execution times of all processes in a system equals the number of
123627 processors, multiplied by the elapsed time, assuming that no processor is idle during that
123628 elapsed time. However, in some implementations it might not be possible to relate CPU
123629 time to elapsed time. For example, in a heterogeneous multi-processor system in which
123630 each processor runs at a different speed, an implementation may choose to define each
123631 “second” of CPU time to be a certain number of “cycles” that a CPU has executed.

123632 Existing Practice

123633 Measuring and limiting the execution time of each concurrent activity are common
123634 features of most industrial implementations of realtime systems. Almost all critical
123635 realtime systems are currently built upon a cyclic executive. With this approach, a regular
123636 timer interrupt kicks off the next sequence of computations. It also checks that the current
123637 sequence has completed. If it has not, then some error recovery action can be undertaken
123638 (or at least an overrun is avoided). Current software engineering principles and the
123639 increasing complexity of software are driving application developers to implement these
123640 systems on multi-threaded or multi-process operating systems. Therefore, if a POSIX
123641 operating system is to be used for this type of application, then it must offer the same level
123642 of protection.

123643 Execution time clocks are also common in most UNIX implementations, although these
123644 clocks usually have requirements different from those of realtime applications. The

123645 POSIX.1 *times()* function supports the measurement of the execution time of the calling
123646 process, and its terminated child processes. This execution time is measured in clock ticks
123647 and is supplied as two different values with the user and system execution times,
123648 respectively. BSD supports the function *getrusage()*, which allows the calling process to get
123649 information about the resources used by itself and/or all of its terminated child processes.
123650 The resource usage includes user and system CPU time. Some UNIX systems have options
123651 to specify high resolution (up to one microsecond) CPU-time clocks using the *times()* or
123652 the *getrusage()* functions.

123653 The *times()* and *getrusage()* interfaces do not meet important realtime requirements, such
123654 as the possibility of monitoring execution time from a different process or thread, or the
123655 possibility of detecting an execution time overrun. The latter requirement is supported in
123656 some UNIX implementations that are able to send a signal when the execution time of a
123657 process has exceeded some specified value. For example, BSD defines the functions
123658 *getitimer()* and *setitimer()*, which can operate either on a realtime clock (wall-clock), or on
123659 virtual-time or profile-time clocks which measure CPU time in two different ways. These
123660 functions do not support access to the execution time of other processes.

123661 The IBM MVS operating system supports per-process and per-thread execution time
123662 clocks. It also supports limiting the execution time of a given process.

123663 Given all this existing practice, the working group considered that the POSIX.1 clocks and
123664 timers interface was appropriate to meet most of the requirements that realtime
123665 applications have for execution time clocks. Functions were added to get the CPU time
123666 clock IDs, and to allow/disallow the thread CPU-time clocks (in order to preserve the
123667 efficiency of some implementations of threads).

123668 Clock Constants

123669 The definition of the manifest constants *CLOCK_PROCESS_CPUTIME_ID* and
123670 *CLOCK_THREAD_CPUTIME_ID* allows processes or threads, respectively, to access their
123671 own execution-time clocks. However, given a process or thread, access to its own
123672 execution-time clock is also possible if the clock ID of this clock is obtained through a call
123673 to *clock_getcpuclockid()* or *pthread_getcpuclockid()*. Therefore, these constants are not
123674 necessary and could be deleted to make the interface simpler. Their existence saves one
123675 system call in the first access to the CPU-time clock of each process or thread. The working
123676 group considered this issue and decided to leave the constants in POSIX.1-2017 because
123677 they are closer to the POSIX.1b use of clock identifiers.

123678 Library Implementations of Threads

123679 In library implementations of threads, kernel entities and library threads can coexist. In
123680 this case, if the CPU-time clocks are supported, most of the clock and timer functions will
123681 need to have two implementations: one in the thread library, and one in the system calls
123682 library. The main difference between these two implementations is that the thread library
123683 implementation will have to deal with clocks and timers that reside in the thread space,
123684 while the kernel implementation will operate on timers and clocks that reside in kernel
123685 space. In the library implementation, if the clock ID refers to a clock that resides in the
123686 kernel, a kernel call will have to be made. The correct version of the function can be chosen
123687 by specifying the appropriate order for the libraries during the link process.

123688 History of Resolution Issues: Deletion of the *enable* Attribute

123689 In early proposals, consideration was given to inclusion of an attribute called *enable* for
123690 CPU-time clocks. This would allow implementations to avoid the overhead of measuring
123691 execution time for those processes or threads for which this measurement was not
123692 required. However, this is unnecessary since processes are already required to measure

123693 execution time by the POSIX.1 *times()* function. Consequently, the *enable* attribute is not
 123694 present.

123695 **Rationale Relating to Timeouts**

123696 Requirements for Timeouts

123697 Realtime systems which must operate reliably over extended periods without human
 123698 intervention are characteristic in embedded applications such as avionics, machine control,
 123699 and space exploration, as well as more mundane applications such as cable TV, security
 123700 systems, and plant automation. A multi-tasking paradigm, in which many independent
 123701 and/or cooperating software functions relinquish the processor(s) while waiting for a
 123702 specific stimulus, resource, condition, or operation completion, is very useful in producing
 123703 well engineered programs for such systems. For such systems to be robust and fault-
 123704 tolerant, expected occurrences that are unduly delayed or that never occur must be
 123705 detected so that appropriate recovery actions may be taken. This is difficult if there is no
 123706 way for a task to regain control of a processor once it has relinquished control (blocked)
 123707 awaiting an occurrence which, perhaps because of corrupted code, hardware malfunction,
 123708 or latent software bugs, will not happen when expected. Therefore, the common practice
 123709 in realtime operating systems is to provide a capability to time out such blocking services.
 123710 Although there are several methods to achieve this already defined by POSIX, none are as
 123711 reliable or efficient as initiating a timeout simultaneously with initiating a blocking service.
 123712 This is especially critical in hard-realtime embedded systems because the processors
 123713 typically have little time reserve, and allowed fault recovery times are measured in
 123714 milliseconds rather than seconds.

123715 The working group largely agreed that such timeouts were necessary and ought to become
 123716 part of POSIX.1-2017, particularly vendors of realtime operating systems whose customers
 123717 had already expressed a strong need for timeouts. There was some resistance to inclusion
 123718 of timeouts in POSIX.1-2017 because the desired effect, fault tolerance, could, in theory, be
 123719 achieved using existing facilities and alternative software designs, but there was no
 123720 compelling evidence that realtime system designers would embrace such designs at the
 123721 sacrifice of performance and/or simplicity.

123722 Which Services should be Timed Out?

123723 Originally, the working group considered the prospect of providing timeouts on all
 123724 blocking services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c,
 123725 and future interfaces to be defined by other working groups, as sort of a general policy.
 123726 This was rather quickly rejected because of the scope of such a change, and the fact that
 123727 many of those services would not normally be used in a realtime context. More traditional
 123728 timesharing solutions to timeout would suffice for most of the POSIX.1 interfaces, while
 123729 others had asynchronous alternatives which, while more complex to utilize, would be
 123730 adequate for some realtime and all non-realtime applications.

123731 The list of potential candidates for timeouts was narrowed to the following for further
 123732 consideration:

123733 ‡ POSIX.1b

123734 ‡ *sem_wait()*

123735 ‡ *mq_receive()*

123736 ‡ *mq_send()*

123737 ‡~~lio_listio()~~
 123738 ‡~~lio_suspend()~~
 123739 ‡~~sigwait()~~ (timeout already implemented by *sigtimedwait()*)
 123740 ‡~~OSX.1c~~
 123741 ‡~~pthread_mutex_lock()~~
 123742 ‡~~pthread_join()~~
 123743 ‡~~pthread_cond_wait()~~
 123744 (timeout already implemented by *pthread_cond_timedwait()*)

123745 ‡~~OSX.1~~

123746 ‡~~read()~~

123747 ‡~~write()~~

123748 After further review by the working group, the *lio_listio()*, *read()*, and *write()* functions (all
 123749 forms of blocking synchronous I/O) were eliminated from the list because of the
 123750 following:

123751 ‡ ~~syn~~ Asynchronous alternatives exist

123752 ‡ ~~int~~ Timeouts can be implemented, albeit non-portably, in device drivers

123753 ‡ ~~str~~ Strong desire not to introduce modifications to POSIX.1 interfaces

123754 The working group ultimately rejected *pthread_join()* since both that interface and a timed
 123755 variant of that interface are non-minimal and may be implemented as a function. See
 123756 below for a library implementation of *pthread_join()*.

123757 Thus, there was a consensus among the working group members to add timeouts to 4 of
 123758 the remaining 5 functions (the timeout for *lio_suspend()* was ultimately added directly to
 123759 POSIX.1b, while the others were added by POSIX.1d). However, *pthread_mutex_lock()*
 123760 remained contentious.

123761 Many feel that *pthread_mutex_lock()* falls into the same class as the other functions; that is,
 123762 it is desirable to time out a mutex lock because a mutex may fail to be unlocked due to
 123763 errant or corrupted code in a critical section (looping or branching outside of the unlock
 123764 code), and therefore is equally in need of a reliable, simple, and efficient timeout. In fact,
 123765 since mutexes are intended to guard small critical sections, most *pthread_mutex_lock()* calls
 123766 would be expected to obtain the lock without blocking nor utilizing any kernel service,
 123767 even in implementations of threads with global contention scope; the timeout alternative
 123768 need only be considered after it is determined that the thread must block.

123769 Those opposed to timing out mutexes feel that the very simplicity of the mutex is
 123770 compromised by adding a timeout semantic, and that to do so is senseless. They claim that
 123771 if a timed mutex is really deemed useful by a particular application, then it can be
 123772 constructed from the facilities already in POSIX.1b and POSIX.1c. The following two C-
 123773 language library implementations of mutex locking with timeout represent the solutions
 123774 offered (in both implementations, the timeout parameter is specified as absolute time, not
 123775 relative time as in the proposed POSIX.1c interfaces).

```

123776      Spinlock Implementation
123777      #include <pthread.h>
123778      #include <time.h>
123779      #include <errno.h>
123780
123780      int pthread_mutex_timedlock(pthread_mutex_t *mutex,
123781                                const struct timespec *timeout)
123782      {
123783          struct timespec timenow;
123784
123784          while (pthread_mutex_trylock(mutex) == EBUSY)
123785              {
123786                  clock_gettime(CLOCK_REALTIME, &timenow);
123787                  if (timespec_cmp(&timenow, timeout) >= 0)
123788                      {
123789                          return ETIMEDOUT;
123790                      }
123791                  pthread_yield();
123792              }
123793          return 0;
123794      }

```

123795 The Spinlock implementation is generally unsuitable for any application using priority-
123796 based thread scheduling policies such as SCHED_FIFO or SCHED_RR, since the mutex
123797 could currently be held by a thread of lower priority within the same allocation domain,
123798 but since the waiting thread never blocks, only threads of equal or higher priority will ever
123799 run, and the mutex cannot be unlocked. Setting priority inheritance or priority ceiling
123800 protocol on the mutex does not solve this problem, since the priority of a mutex owning
123801 thread is only boosted if higher priority threads are blocked waiting for the mutex; clearly
123802 not the case for this spinlock.

123803 Condition Wait Implementation

```

123804      #include <pthread.h>
123805      #include <time.h>
123806      #include <errno.h>
123807
123807      struct timed_mutex
123808      {
123809          int locked;
123810          pthread_mutex_t mutex;
123811          pthread_cond_t cond;
123812      };
123813      typedef struct timed_mutex timed_mutex_t;
123814
123814      int timed_mutex_lock(timed_mutex_t *tm,
123815                          const struct timespec *timeout)
123816      {
123817          int timedout=FALSE;
123818          int error_status;
123819
123819          pthread_mutex_lock(&tm->mutex);
123820
123820          while (tm->locked && !timedout)
123821              {
123822                  if ((error_status=pthread_cond_timedwait(&tm->cond,
123823                                                         &tm->mutex,

```

```

123824         timeout))!=0)
123825     {
123826         if (error_status==ETIMEDOUT) timedout = TRUE;
123827     }
123828 }
123829
123829     if(timedout)
123830     {
123831         pthread_mutex_unlock(&tm->mutex);
123832         return ETIMEDOUT;
123833     }
123834     else
123835     {
123836         tm->locked = TRUE;
123837         pthread_mutex_unlock(&tm->mutex);
123838         return 0;
123839     }
123840 }
123841
123841 void timed_mutex_unlock(timed_mutex_t *tm)
123842 {
123843     pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
123844     tm->locked = FALSE;
123845     pthread_mutex_unlock(&tm->mutex);
123846     pthread_cond_signal(&tm->cond);
123847 }

```

123848 The Condition Wait implementation effectively substitutes the *pthread_cond_timedwait()*
123849 function (which is currently timed out) for the desired *pthread_mutex_timedlock()*. Since
123850 waits on condition variables currently do not include protocols which avoid priority
123851 inversion, this method is generally unsuitable for realtime applications because it does not
123852 provide the same priority inversion protection as the untimed *pthread_mutex_lock()*. Also,
123853 for any given implementations of the current mutex and condition variable primitives, this
123854 library implementation has a performance cost at least 2.5 times that of the untimed
123855 *pthread_mutex_lock()* even in the case where the timed mutex is readily locked without
123856 blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or
123857 where assignment is atomic, at least an additional *pthread_cond_signal()* is required.
123858 *pthread_mutex_timedlock()* could be implemented at effectively no performance penalty in
123859 this case because the timeout parameters need only be considered after it is determined
123860 that the mutex cannot be locked immediately.

123861 Thus it has not yet been shown that the full semantics of mutex locking with timeout can
123862 be efficiently and reliably achieved using existing interfaces. Even if the existence of an
123863 acceptable library implementation were proven, it is difficult to justify why the interface
123864 itself should not be made portable, especially considering approval for the other four
123865 timeouts.

123866 Rationale for Library Implementation of *pthread_timedjoin()*

123867 Library implementation of *pthread_timedjoin()*:

```

123868 /*
123869  * Construct a thread variety entirely from existing functions
123870  * with which a join can be done, allowing the join to time out.
123871  */
123872 #include <pthread.h>

```

```

123873     #include <time.h>
123874     struct timed_thread {
123875         pthread_t t;
123876         pthread_mutex_t m;
123877         int exiting;
123878         pthread_cond_t exit_c;
123879         void *(*start_routine)(void *arg);
123880         void *arg;
123881         void *status;
123882     };
123883     typedef struct timed_thread *timed_thread_t;
123884     static pthread_key_t timed_thread_key;
123885     static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;
123886     static void timed_thread_init()
123887     {
123888         pthread_key_create(&timed_thread_key, NULL);
123889     }
123890     static void *timed_thread_start_routine(void *args)
123891     /*
123892      * Routine to establish thread-specific data value and run the actual
123893      * thread start routine which was supplied to timed_thread_create().
123894      */
123895     {
123896         timed_thread_t tt = (timed_thread_t) args;
123897         pthread_once(&timed_thread_once, timed_thread_init);
123898         pthread_setspecific(timed_thread_key, (void *)tt);
123899         timed_thread_exit((tt->start_routine)(tt->arg));
123900     }
123901     int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
123902         void *(*start_routine)(void *), void *arg)
123903     /*
123904      * Allocate a thread which can be used with timed_thread_join().
123905      */
123906     {
123907         timed_thread_t tt;
123908         int result;
123909         tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
123910         pthread_mutex_init(&tt->m, NULL);
123911         tt->exiting = FALSE;
123912         pthread_cond_init(&tt->exit_c, NULL);
123913         tt->start_routine = start_routine;
123914         tt->arg = arg;
123915         tt->status = NULL;
123916         if ((result = pthread_create(&tt->t, attr,
123917             timed_thread_start_routine, (void *)tt)) != 0) {
123918             free(tt);
123919             return result;
123920         }

```

```

123921         pthread_detach(tt->t);
123922         ttp = tt;
123923         return 0;
123924     }

123925     int timed_thread_join(timed_thread_t tt,
123926                          struct timespec *timeout,
123927                          void **status)
123928     {
123929         int result;

123930         pthread_mutex_lock(&tt->m);
123931         result = 0;
123932         /*
123933          * Wait until the thread announces that it is exiting,
123934          * or until timeout.
123935          */
123936         while (result == 0 && ! tt->exiting) {
123937             result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
123938         }
123939         pthread_mutex_unlock(&tt->m);
123940         if (result == 0 && tt->exiting) {
123941             *status = tt->status;
123942             free((void *)tt);
123943             return result;
123944         }
123945         return result;
123946     }

123947     void timed_thread_exit(void *status)
123948     {
123949         timed_thread_t tt;
123950         void *specific;

123951         if ((specific=pthread_getspecific(timed_thread_key)) == NULL){
123952             /*
123953              * Handle cases which will not happen with correct usage.
123954              */
123955             pthread_exit( NULL);
123956         }
123957         tt = (timed_thread_t) specific;
123958         pthread_mutex_lock(&tt->m);
123959         /*
123960          * Tell a joiner that we are exiting.
123961          */
123962         tt->status = status;
123963         tt->exiting = TRUE;
123964         pthread_cond_signal(&tt->exit_c);
123965         pthread_mutex_unlock(&tt->m);
123966         /*
123967          * Call pthread exit() to call destructors and really
123968          * exit the thread.
123969          */
123970         pthread_exit(NULL);
123971     }

```

123972 The `pthread_join()` C-language example shown above demonstrates that it is possible,
123973 using existing pthread facilities, to construct a variety of thread which allows for joining
123974 such a thread, but which allows the join operation to time out. It does this by using a
123975 `pthread_cond_timedwait()` to wait for the thread to exit. A **timed_thread_t** descriptor
123976 structure is used to pass parameters from the creating thread to the created thread, and
123977 from the exiting thread to the joining thread. This implementation is roughly equivalent to
123978 what a normal `pthread_join()` implementation would do, with the single change being that
123979 `pthread_cond_timedwait()` is used in place of a simple `pthread_cond_wait()`.

123980 Since it is possible to implement such a facility entirely from existing pthread interfaces,
123981 and with roughly equal efficiency and complexity to an implementation which would be
123982 provided directly by a pthreads implementation, it was the consensus of the working
123983 group members that any `pthread_timedjoin()` facility would be unnecessary, and should not
123984 be provided.

123985 Form of the Timeout Interfaces

123986 The working group considered a number of alternative ways to add timeouts to blocking
123987 services. At first, a system interface which would specify a one-shot or persistent timeout
123988 to be applied to subsequent blocking services invoked by the calling process or thread was
123989 considered because it allowed all blocking services to be timed out in a uniform manner
123990 with a single additional interface; this was rather quickly rejected because it could easily
123991 result in the wrong services being timed out.

123992 It was suggested that a timeout value might be specified as an attribute of the object
123993 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either
123994 on a case-by-case basis or for all timeouts.

123995 Looking at the two existing timeouts for blocking services indicates that the working
123996 group members favor a separate interface for the timed version of a function. However,
123997 `pthread_cond_timedwait()` utilizes an absolute timeout value while `sigtimedwait()` uses a
123998 relative timeout value. The working group members agreed that relative timeout values
123999 are appropriate where the timeout mechanism's primary use was to deal with an
124000 unexpected or error situation, but they are inappropriate when the timeout must expire at
124001 a particular time, or before a specific deadline. For the timeouts being introduced in
124002 POSIX.1-2017, the working group considered allowing both relative and absolute timeouts
124003 as is done with POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

124004 An absolute time measure can be easily implemented on top of an interface that specifies
124005 relative time, by reading the clock, calculating the difference between the current time and
124006 the desired wakeup time, and issuing a relative timeout call. But there is a race condition
124007 with this approach because the thread could be preempted after reading the clock, but
124008 before making the timed-out call; in this case, the thread would be awakened later than it
124009 should and, thus, if the wakeup time represented a deadline, it would miss it.

124010 There is also a race condition when trying to build a relative timeout on top of an interface
124011 that specifies absolute timeouts. In this case, the clock would have to be read to calculate
124012 the absolute wakeup time as the sum of the current time plus the relative timeout interval.
124013 In this case, if the thread is preempted after reading the clock but before making the timed-
124014 out call, the thread would be awakened earlier than desired.

124015 But the race condition with the absolute timeouts interface is not as bad as the one that
124016 happens with the relative timeout interface, because there are simple workarounds. For the
124017 absolute timeouts interface, if the timing requirement is a deadline, the deadline can still
124018 be met because the thread woke up earlier than the deadline. If the timeout is just used as
124019 an error recovery mechanism, the precision of timing is not really important. If the timing
124020 requirement is that between actions A and B a minimum interval of time must elapse, the

124021 absolute timeout interface can be safely used by reading the clock after action A has been
124022 started. It could be argued that, since the call with the absolute timeout is atomic from the
124023 application point of view, it is not possible to read the clock after action A, if this action is
124024 part of the timed-out call. But looking at the nature of the calls for which timeouts are
124025 specified (locking a mutex, waiting for a semaphore, waiting for a message, or waiting
124026 until there is space in a message queue), the timeouts that an application would build on
124027 these actions would not be triggered by these actions themselves, but by some other
124028 external action. For example, if waiting for a message to arrive to a message queue, and
124029 waiting for at least 20 milliseconds, this time interval would start to be counted from some
124030 event that would trigger both the action that produces the message, as well as the action
124031 that waits for the message to arrive, and not by the wait-for-message operation itself. In
124032 this case, the workaround proposed above could be used.

124033 For these reasons, the absolute timeout is preferred over the relative timeout interface.

124034 B.2.9 Threads

124035 Threads will normally be more expensive than subroutines (or functions, routines, and so on) if
124036 specialized hardware support is not provided. Nevertheless, threads should be sufficiently
124037 efficient to encourage their use as a medium to fine-grained structuring mechanism for
124038 parallelism in an application. Structuring an application using threads then allows it to take
124039 immediate advantage of any underlying parallelism available in the host environment. This
124040 means implementors are encouraged to optimize for fast execution at the possible expense of
124041 efficient utilization of storage. For example, a common thread creation technique is to cache
124042 appropriate thread data structures. That is, rather than releasing system resources, the
124043 implementation retains these resources and reuses them when the program next asks to create a
124044 new thread. If this reuse of thread resources is to be possible, there has to be very little unique
124045 state associated with each thread, because any such state has to be reset when the thread is
124046 reused.

124047 Thread Creation Attributes

124048 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to
124049 support probable future standardization in these areas without requiring that the interface itself
124050 be changed.

124051 Attributes objects provide clean isolation of the configurable aspects of threads. For example,
124052 ``stack size'' is an important attribute of a thread, but it cannot be expressed portably. When
124053 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects
124054 can help by allowing the changes to be isolated in a single place, rather than being spread across
124055 every instance of thread creation.

124056 Attributes objects can be used to set up *classes* of threads with similar attributes; for example,
124057 ``threads with large stacks and high priority'' or ``threads with minimal stacks''. These classes
124058 can be defined in a single place and then referenced wherever threads need to be created.
124059 Changes to ``class'' decisions become straightforward, and detailed analysis of each
124060 *pthread_create()* call is not required.

124061 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had
124062 been specified as structures, adding new attributes would force recompilation of all multi-
124063 threaded programs when the attributes objects are extended; this might not be possible if
124064 different program components were supplied by different vendors.

124065 Additionally, opaque attributes objects present opportunities for improving performance.
124066 Argument validity can be checked once when attributes are set, rather than each time a thread is

124067 created. Implementations will often need to cache kernel objects that are expensive to create.
124068 Opaque attributes objects provide an efficient mechanism to detect when cached objects become
124069 invalid due to attribute changes.

124070 Because assignment is not necessarily defined on a given opaque type, implementation-defined
124071 default values cannot be defined in a portable way. The solution to this problem is to allow
124072 attribute objects to be initialized dynamically by attributes object initialization functions, so that
124073 default values can be supplied automatically by the implementation.

124074 The following proposal was provided as a suggested alternative to the supplied attributes:

- 124075 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to
124076 the initialization routines (*pthread_create()*, *pthread_mutex_init()*, *pthread_cond_init()*). The
124077 parameter containing the flags should be an opaque type for extensibility. If no flags are
124078 set in the parameter, then the objects are created with default characteristics. An
124079 implementation may specify implementation-defined flag values and associated
124080 behavior.
- 124081 2. If further specialization of mutexes and condition variables is necessary, implementations
124082 may specify additional procedures that operate on the **pthread_mutex_t** and
124083 **pthread_cond_t** objects (instead of on attributes objects).

124084 The difficulties with this solution are:

- 124085 1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using
124086 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,
124087 application programmers need to know the location of each bit. If bits are set or read by
124088 encapsulation (that is, *get*()* or *set*()* functions), then the bitmask is merely an
124089 implementation of attributes objects as currently defined and should not be exposed to
124090 the programmer.
- 124091 2. Many attributes are not Boolean or very small integral values. For example, scheduling
124092 policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking
124093 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,
124094 the bitmask can only reasonably control whether particular attributes are set or not, and it
124095 cannot serve as the repository of the value itself. The value needs to be specified as a
124096 function parameter (which is non-extensible), or by setting a structure field (which is non-
124097 opaque), or by *get*()* and *set*()* functions (making the bitmask a redundant addition to
124098 the attributes objects).

124099 Stack size is defined as an optional attribute because the very notion of a stack is inherently
124100 machine-dependent. Some implementations may not be able to change the size of the stack, for
124101 example, and others may not need to because stack pages may be discontinuous and can be
124102 allocated and released on demand.

124103 The attribute mechanism has been designed in large measure for extensibility. Future extensions
124104 to the attribute mechanism or to any attributes object defined in POSIX.1-2017 have to be done
124105 with care so as not to affect binary-compatibility.

124106 Attribute objects, even if allocated by means of dynamic allocation functions such as *malloc()*,
124107 may have their size fixed at compile time. This means, for example, a *pthread_create()* in an
124108 implementation with extensions to the **pthread_attr_t** cannot look beyond the area that the
124109 binary application assumes is valid. This suggests that implementations should maintain a size
124110 field in the attributes object, as well as possibly version information, if extensions in different
124111 directions (possibly by different vendors) are to be accommodated.

124112 Thread Implementation Models

124113 There are various thread implementation models. At one end of the spectrum is the “library-
124114 thread model”. In such a model, the threads of a process are not visible to the operating system
124115 kernel, and the threads are not kernel-scheduled entities. The process is the only kernel-
124116 scheduled entity. The process is scheduled onto the processor by the kernel according to the
124117 scheduling attributes of the process. The threads are scheduled onto the single kernel-scheduled
124118 entity (the process) by the runtime library according to the scheduling attributes of the threads.
124119 A problem with this model is that it constrains concurrency. Since there is only one kernel-
124120 scheduled entity (namely, the process), only one thread per process can execute at a time. If the
124121 thread that is executing blocks on I/O, then the whole process blocks.

124122 At the other end of the spectrum is the “kernel-thread model”. In this model, all threads are
124123 visible to the operating system kernel. Thus, all threads are kernel-scheduled entities, and all
124124 threads can concurrently execute. The threads are scheduled onto processors by the kernel
124125 according to the scheduling attributes of the threads. The drawback to this model is that the
124126 creation and management of the threads entails operating system calls, as opposed to subroutine
124127 calls, which makes kernel threads heavier weight than library threads.

124128 Hybrids of these two models are common. A hybrid model offers the speed of library threads
124129 and the concurrency of kernel threads. In hybrid models, a process has some (relatively small)
124130 number of kernel scheduled entities associated with it. It also has a potentially much larger
124131 number of library threads associated with it. Some library threads may be bound to kernel-
124132 scheduled entities, while the other library threads are multiplexed onto the remaining kernel-
124133 scheduled entities. There are two levels of thread scheduling:

- 124134 1. The runtime library manages the scheduling of (unbound) library threads onto kernel-
124135 scheduled entities.
- 124136 2. The kernel manages the scheduling of kernel-scheduled entities onto processors.

124137 For this reason, a hybrid model is referred to as a two-level threads scheduling model. In this
124138 model, the process can have multiple concurrently executing threads; specifically, it can have as
124139 many concurrently executing threads as it has kernel-scheduled entities.

124140 Thread-Specific Data

124141 Many applications require that a certain amount of context be maintained on a per-thread basis
124142 across procedure calls. A common example is a multi-threaded library routine that allocates
124143 resources from a common pool and maintains an active resource list for each thread. The thread-
124144 specific data interface provided to meet these needs may be viewed as a two-dimensional array
124145 of values with keys serving as the row index and thread IDs as the column index (although the
124146 implementation need not work this way).

124147 Models

124148 Three possible thread-specific data models were considered:

- 124149 1. No Explicit Support

124150 A standard thread-specific data interface is not strictly necessary to support
124151 applications that require per-thread context. One could, for example, provide a hash
124152 function that converted a `pthread_t` into an integer value that could then be used to
124153 index into a global array of per-thread data pointers. This hash function, in
124154 conjunction with `pthread_self()`, would be all the interface required to support a
124155 mechanism of this sort. Unfortunately, this technique is cumbersome. It can lead to
124156 duplicated code as each set of cooperating modules implements their own per-
124157 thread data management schemes. This technique would also require that `pthread_t`

124158 not be an opaque type.

124159 2. Single (**void ***) Pointer

124160 Another technique would be to provide a single word of per-thread storage and a
 124161 pair of functions to fetch and store the value of this word. The word could then hold
 124162 a pointer to a block of per-thread memory. The allocation, partitioning, and general
 124163 use of this memory would be entirely up to the application. Although this method
 124164 is not as problematic as technique 1, it suffers from interoperability problems. For
 124165 example, all modules using the per-thread pointer would have to agree on a
 124166 common usage protocol.

124167 3. Key/Value Mechanism

124168 This method associates an opaque key (for example, stored in a variable of type
 124169 **pthread_key_t**) with each per-thread datum. These keys play the role of identifiers
 124170 for per-thread data. This technique is the most generic and avoids the problems
 124171 noted above, albeit at the cost of some complexity.

124172 The primary advantage of the third model is its information hiding properties. Modules
 124173 using this model are free to create and use their own key(s) independent of all other such
 124174 usage, whereas the other models require that all modules that use thread-specific context
 124175 explicitly cooperate with all other such modules. The data-independence provided by the
 124176 third model is worth the additional interface. Therefore, the third model was chosen.

124177 Requirements

124178 It is important that it be possible to implement the thread-specific data interface without
 124179 the use of thread private memory. To do otherwise would increase the weight of each
 124180 thread, thereby limiting the range of applications for which the threads interfaces provided
 124181 by POSIX.1-2017 is appropriate.

124182 The values that one binds to the key via *pthread_setspecific()* may, in fact, be pointers to
 124183 shared storage locations available to all threads. It is only the key/value bindings that are
 124184 maintained on a per-thread basis, and these can be kept in any portion of the address space
 124185 that is reserved for use by the calling thread (for example, on the stack). Thus, no per-
 124186 thread MMU state is required to implement the interface. On the other hand, there is
 124187 nothing in the interface specification to preclude the use of a per-thread MMU state if it is
 124188 available (for example, the key values returned by *pthread_key_create()* could be thread
 124189 private memory addresses).

124190 Standardization Issues

124191 Thread-specific data is a requirement for a usable thread interface. The binding described
 124192 in this section provides a portable thread-specific data mechanism for languages that do
 124193 not directly support a thread-specific storage class. A binding to POSIX.1-2017 for a
 124194 language that does include such a storage class need not provide this specific interface.

124195 If a language were to include the notion of thread-specific storage, it would be desirable
 124196 (but *not* required) to provide an implementation of the pthreads thread-specific data
 124197 interface based on the language feature. For example, assume that a compiler for a C-like
 124198 language supports a *private* storage class that provides thread-specific storage. Something
 124199 similar to the following macros might be used to effect a compatible implementation:

```
124200 #define pthread_key_t           private void *
124201 #define pthread_key_create(key) /* no-op */
124202 #define pthread_setspecific(key,value) (key)=(value)
124203 #define pthread_getspecific(key)     (key)
```

124204 **Note:** For the sake of clarity, this example ignores destructor functions. A correct
124205 implementation would have to support them.

124206 **Barriers**

124207 Background

124208 Barriers are typically used in parallel DO/FOR loops to ensure that all threads have
124209 reached a particular stage in a parallel computation before allowing any to proceed to the
124210 next stage. Highly efficient implementation is possible on machines which support a
124211 “Fetch and Add” operation as described in the referenced Almasi and Gottlieb (1989).

124212 The use of return value PTHREAD_BARRIER_SERIAL_THREAD is shown in the
124213 following example:

```
124214 if ( (status=pthread_barrier_wait(&barrier)) ==
124215     PTHREAD_BARRIER_SERIAL_THREAD) {
124216     ...serial section
124217 }
124218     else if (status != 0) {
124219     ...error processing
124220 }
124221 status=pthread_barrier_wait(&barrier);
124222 ...
```

124223 This behavior allows a serial section of code to be executed by one thread as soon as all
124224 threads reach the first barrier. The second barrier prevents the other threads from
124225 proceeding until the serial section being executed by the one thread has completed.

124226 Although barriers can be implemented with mutexes and condition variables, the
124227 referenced Almasi and Gottlieb (1989) provides ample illustration that such
124228 implementations are significantly less efficient than is possible. While the relative
124229 efficiency of barriers may well vary by implementation, it is important that they be
124230 recognized in the POSIX.1-2017 to facilitate applications portability while providing the
124231 necessary freedom to implementors.

124232 Lack of Timeout Feature

124233 Alternate versions of most blocking routines have been provided to support watchdog
124234 timeouts. No alternate interface of this sort has been provided for barrier waits for the
124235 following reasons:

124236 Multiple threads may use different timeout values, some of which may be indefinite.
124237 It is not clear which threads should break through the barrier with a timeout error if
124238 and when these timeouts expire.

124239 The barrier may become unusable once a thread breaks out of a *pthread_barrier_wait()*
124240 with a timeout error. There is, in general, no way to guarantee the consistency of a
124241 barrier’s internal data structures once a thread has timed out of a
124242 *pthread_barrier_wait()*. Even the inclusion of a special barrier reinitialization function
124243 would not help much since it is not clear how this function would affect the behavior
124244 of threads that reach the barrier between the original timeout and the call to the
124245 reinitialization function.

124246 **Spin Locks**

124247 Background

124248 Spin locks represent an extremely low-level synchronization mechanism suitable primarily
 124249 for use on shared memory multi-processors. It is typically an atomically modified Boolean
 124250 value that is set to one when the lock is held and to zero when the lock is freed.

124251 When a caller requests a spin lock that is already held, it typically spins in a loop testing
 124252 whether the lock has become available. Such spinning wastes processor cycles so the lock
 124253 should only be held for short durations and not across sleep/block operations. Callers
 124254 should unlock spin locks before calling sleep operations.

124255 Spin locks are available on a variety of systems. The functions included in POSIX.1-2017
 124256 are an attempt to standardize that existing practice.

124257 Lack of Timeout Feature

124258 Alternate versions of most blocking routines have been provided to support watchdog
 124259 timeouts. No alternate interface of this sort has been provided for spin locks for the
 124260 following reasons:

124261 It is impossible to determine appropriate timeout intervals for spin locks in a
 124262 portable manner. The amount of time one can expect to spend spin-waiting is
 124263 inversely proportional to the degree of parallelism provided by the system.

124264 It can vary from a few cycles when each competing thread is running on its own
 124265 processor, to an indefinite amount of time when all threads are multiplexed on a
 124266 single processor (which is why spin locking is not advisable on uniprocessors).

124267 When used properly, the amount of time the calling thread spends waiting on a spin
 124268 lock should be considerably less than the time required to set up a corresponding
 124269 watchdog timer. Since the primary purpose of spin locks is to provide a low-
 124270 overhead synchronization mechanism for multi-processors, the overhead of a
 124271 timeout mechanism was deemed unacceptable.

124272 It was also suggested that an additional *count* argument be provided (on the
 124273 *pthread_spin_lock()* call) in lieu of a true timeout so that a spin lock call could fail gracefully
 124274 if it was unable to apply the lock after *count* attempts. This idea was rejected because it is
 124275 not existing practice. Furthermore, the same effect can be obtained with
 124276 *pthread_spin_trylock()*, as illustrated below:

```

124277 int n = MAX_SPIN;
124278 while ( --n >= 0 )
124279 {
124280     if ( !pthread_spin_try_lock(...) )
124281         break;
124282 }
124283 if ( n >= 0 )
124284 {
124285     /* Successfully acquired the lock */
124286 }
124287 else
124288 {
124289     /* Unable to acquire the lock */
124290 }

```

124291 *process-shared* Attribute

124292 The initialization functions associated with most POSIX synchronization objects (for
124293 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-*
124294 *shared* attribute that specifies whether or not the object is to be shared across processes. In
124295 the draft corresponding to the first balloting round, two separate initialization functions
124296 are provided for spin locks, however: one for spin locks that were to be shared across
124297 processes (*spin_init()*), and one for locks that were only used by multiple threads within a
124298 single process (*pthread_spin_init()*). This was done so as to keep the overhead associated
124299 with spin waiting to an absolute minimum. However, the balloting group requested that,
124300 since the overhead associated to a bit check was small, spin locks should be consistent with
124301 the rest of the synchronization primitives, and thus the *process-shared* attribute was
124302 introduced for spin locks.

124303 Spin Locks *versus* Mutexes

124304 It has been suggested that mutexes are an adequate synchronization mechanism and spin
124305 locks are not necessary. Locking mechanisms typically must trade off the processor
124306 resources consumed while setting up to block the thread and the processor resources
124307 consumed by the thread while it is blocked. Spin locks require very little resources to set
124308 up the blocking of a thread. Existing practice is to simply loop, repeating the atomic
124309 locking operation until the lock is available. While the resources consumed to set up
124310 blocking of the thread are low, the thread continues to consume processor resources while
124311 it is waiting.

124312 On the other hand, mutexes may be implemented such that the processor resources
124313 consumed to block the thread are large relative to a spin lock. After detecting that the
124314 mutex lock is not available, the thread must alter its scheduling state, add itself to a set of
124315 waiting threads, and, when the lock becomes available again, undo all of this before taking
124316 over ownership of the mutex. However, while a thread is blocked by a mutex, no processor
124317 resources are consumed.

124318 Therefore, spin locks and mutexes may be implemented to have different characteristics.
124319 Spin locks may have lower overall overhead for very short-term blocking, and mutexes
124320 may have lower overall overhead when a thread will be blocked for longer periods of time.
124321 The presence of both interfaces allows implementations with these two different
124322 characteristics, both of which may be useful to a particular application.

124323 It has also been suggested that applications can build their own spin locks from the
124324 *pthread_mutex_trylock()* function:

```
124325 while (pthread_mutex_trylock(&mutex));
```

124326 The apparent simplicity of this construct is somewhat deceiving, however. While the actual
124327 wait is quite efficient, various guarantees on the integrity of mutex objects (for example,
124328 priority inheritance rules) may add overhead to the successful path of the trylock
124329 operation that is not required of spin locks. One could, of course, add an attribute to the
124330 mutex to bypass such overhead, but the very act of finding and testing this attribute
124331 represents more overhead than is found in the typical spin lock.

124332 The need to hold spin lock overhead to an absolute minimum also makes it impossible to
124333 provide guarantees against starvation similar to those provided for mutexes or read-write
124334 locks. The overhead required to implement such guarantees (for example, disabling
124335 preemption before spinning) may well exceed the overhead of the spin wait itself by many
124336 orders of magnitude. If a "safe" spin wait seems desirable, it can always be provided
124337 (albeit at some performance cost) via appropriate mutex attributes.

124338 Robust Mutexes

124339 Robust mutexes are intended to protect applications that use mutexes to protect data shared
124340 between different processes. If a process is terminated by a signal while a thread is holding a
124341 mutex, there is no chance for the process to clean up after it. Waiters for the locked mutex might
124342 wait indefinitely.

124343 With robust mutexes the problem can be solved: whenever a fatal signal terminates a process,
124344 current or future waiters of the mutex are notified about this fact. The locking function provides
124345 notification of this condition through the error condition [EOWNERDEAD]. A thread then has
124346 the chance to clean up the state protected by the mutex and mark the state as consistent again by
124347 a call to `pthread_mutex_consistent()`.

124348 Pre-existing implementations have used the semantics of robust mutexes for a variety of
124349 situations, some of them not defined in the standard. Where a normally terminated process (i.e.,
124350 when one thread calls `exit()`) causes notification of other waiters of robust mutexes if the mutex
124351 is locked by any thread in the process. This behavior is defined in the standard and makes sense
124352 because no thread other than the thread calling `exit()` has the chance to clean up its data.

124353 If a thread is terminated by cancellation or if it calls `pthread_exit()`, the situation is different. In
124354 both these situations the thread has the chance to clean up after itself by registering appropriate
124355 cleanup handlers. There is no real reason to demand that other waiters for a robust mutex the
124356 terminating thread owns are notified. The committee felt that this is actively encouraging bad
124357 practice because programmers are tempted to rely on the robust mutex semantics instead of
124358 correctly cleaning up after themselves.

124359 Therefore, the standard does not require notification of other waiters at the time a thread is
124360 terminated while the process continues to run. The mutex is still recognized as being locked by
124361 the process (with the thread gone it makes no sense to refer to the thread owning the mutex).
124362 Therefore, a terminating process will cause notifications about the dead owner to be sent to all
124363 waiters. This delay in the notification is not required, but programmers cannot rely on prompt
124364 notification after a thread is terminated.

124365 For the same reason is it not required that an implementation supports robust mutexes that are
124366 not shared between processes. If a robust mutex is used only within one process, all the cleanup
124367 can be performed by the threads themselves by registering appropriate cleanup handlers. Fatal
124368 signals are of no importance in this case because after the signal is delivered there is no thread
124369 remaining to use the mutex.

124370 Some implementations might choose to support intra-process robust mutexes and they might
124371 also send notification of a dead owner right after the previous owner died. But applications
124372 must not rely on this. Applications should only use robust mutexes for the purpose of handling
124373 fatal signals in situations where inter-process mutexes are in use.

124374 Supported Threads Functions

124375 On POSIX-conforming systems, the following symbolic constants are always conforming:

124376 `_POSIX_READER_WRITER_LOCKS`
124377 `_POSIX_THREADS`

124378 Therefore, the following threads functions are always supported:

124379	<i>pthread_atfork()</i>	<i>pthread_mutex_destroy()</i>
124380	<i>pthread_attr_destroy()</i>	<i>pthread_mutex_init()</i>
124381	<i>pthread_attr_getdetachstate()</i>	<i>pthread_mutex_lock()</i>
124382	<i>pthread_attr_getguardsize()</i>	<i>pthread_mutex_trylock()</i>
124383	<i>pthread_attr_getschedparam()</i>	<i>pthread_mutex_unlock()</i>
124384	<i>pthread_attr_init()</i>	<i>pthread_mutexattr_destroy()</i>
124385	<i>pthread_attr_setdetachstate()</i>	<i>pthread_mutexattr_getpshared()</i>
124386	<i>pthread_attr_setguardsize()</i>	<i>pthread_mutexattr_gettype()</i>
124387	<i>pthread_attr_setschedparam()</i>	<i>pthread_mutexattr_init()</i>
124388	<i>pthread_cancel()</i>	<i>pthread_mutexattr_setpshared()</i>
124389	<i>pthread_cleanup_pop()</i>	<i>pthread_mutexattr_settype()</i>
124390	<i>pthread_cleanup_push()</i>	<i>pthread_once()</i>
124391	<i>pthread_cond_broadcast()</i>	<i>pthread_rwlock_destroy()</i>
124392	<i>pthread_cond_destroy()</i>	<i>pthread_rwlock_init()</i>
124393	<i>pthread_cond_init()</i>	<i>pthread_rwlock_rdlock()</i>
124394	<i>pthread_cond_signal()</i>	<i>pthread_rwlock_tryrdlock()</i>
124395	<i>pthread_cond_timedwait()</i>	<i>pthread_rwlock_trywrlock()</i>
124396	<i>pthread_cond_wait()</i>	<i>pthread_rwlock_unlock()</i>
124397	<i>pthread_condattr_destroy()</i>	<i>pthread_rwlock_wrlock()</i>
124398	<i>pthread_condattr_getpshared()</i>	<i>pthread_rwlockattr_destroy()</i>
124399	<i>pthread_condattr_init()</i>	<i>pthread_rwlockattr_getpshared()</i>
124400	<i>pthread_condattr_setpshared()</i>	<i>pthread_rwlockattr_init()</i>
124401	<i>pthread_create()</i>	<i>pthread_rwlockattr_setpshared()</i>
124402	<i>pthread_detach()</i>	<i>pthread_self()</i>
124403	<i>pthread_equal()</i>	<i>pthread_setcancelstate()</i>
124404	<i>pthread_exit()</i>	<i>pthread_setcanceltype()</i>
124405	<i>pthread_getconcurrency()</i>	<i>pthread_setconcurrency()</i>
124406	<i>pthread_getspecific()</i>	<i>pthread_setspecific()</i>
124407	<i>pthread_join()</i>	<i>pthread_sigmask()</i>
124408	<i>pthread_key_create()</i>	<i>pthread_testcancel()</i>
124409	<i>pthread_key_delete()</i>	<i>sigwait()</i>
124410	<i>pthread_kill()</i>	

124411 On POSIX-conforming systems, the symbolic constant `_POSIX_THREAD_SAFE_FUNCTIONS` is
 124412 always defined. Therefore, the following functions are always supported:

124413	<code>asctime_r()</code>	<code>getpwuid_r()</code>
124414	<code>ctime_r()</code>	<code>gmtime_r()</code>
124415	<code>flockfile()</code>	<code>localtime_r()</code>
124416	<code>ftrylockfile()</code>	<code>putc_unlocked()</code>
124417	<code>funlockfile()</code>	<code>putchar_unlocked()</code>
124418	<code>getc_unlocked()</code>	<code>rand_r()</code>
124419	<code>getchar_unlocked()</code>	<code>readdir_r()</code>
124420	<code>getgrgid_r()</code>	<code>strerror_r()</code>
124421	<code>getgrnam_r()</code>	<code>strtok_r()</code>
124422	<code>getpwnam_r()</code>	

124423 Threads Extensions

124424 The following extensions to the IEEE P1003.1c draft standard are now supported in
 124425 POSIX.1-2017 as part of the alignment with the Single UNIX Specification:

124426 Extended mutex attribute types

124427 Read-write locks and attributes (also introduced by the IEEE Std 1003.1j-2000 amendment)

124428 Thread concurrency level

124429 Thread stack guard size

124430 Parallel I/O

124431 Robust mutexes

124432 These extensions carefully follow the threads programming model specified in POSIX.1c. As
 124433 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is
 124434 returned to indicate the error.

124435 The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend
 124436 POSIX.1-2017 without changing the existing interfaces. Attribute objects were defined for
 124437 threads, mutexes, and condition variables. Attributes objects are defined as implementation-
 124438 defined opaque types to aid extensibility, and functions are defined to allow attributes to be set
 124439 or retrieved. This model has been followed when adding the new type attribute of
 124440 **pthread_mutexattr_t** or the new read-write lock attributes object **pthread_rwlockattr_t**.

124441 Extended Mutex Attributes

124442 POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of
 124443 type **pthread_mutexattr_t**, and specifies a number of attributes which this object must
 124444 have and a number of functions which manipulate these attributes. These attributes
 124445 include *detachstate*, *inheritsched*, *schedparm*, *schedpolicy*, *contentionscope*, *stackaddr*, and
 124446 *stacksize*.

124447 The System Interfaces volume of POSIX.1-2017 specifies another mutex attribute called
 124448 *type*. The *type* attribute allows applications to specify the behavior of mutex locking
 124449 operations in situations where POSIX.1c behavior is undefined. The OSF DCE threads
 124450 implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the
 124451 names of the attributes have changed somewhat from the OSF DCE threads
 124452 implementation.

124453 The System Interfaces volume of POSIX.1-2017 also extends the specification of the
 124454 following POSIX.1c functions which manipulate mutexes:

124455 `pthread_mutex_lock()`
 124456 `pthread_mutex_trylock()`
 124457 `pthread_mutex_unlock()`

124458 to take account of the new mutex attribute type and to specify behavior which was
 124459 declared as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now
 124460 depends upon the mutex *type* attribute.

124461 The *type* attribute can have the following values:

124462 PTHREAD_MUTEX_NORMAL

124463 Basic mutex with no specific error checking built in. Does not report a deadlock error.

124464 PTHREAD_MUTEX_RECURSIVE

124465 Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal
 124466 number of times to release the mutex.

124467 PTHREAD_MUTEX_ERRORCHECK

124468 Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is
 124469 not locked by the calling thread or that is not locked at all, or an attempt to relock a
 124470 mutex the thread already owns.

124471 PTHREAD_MUTEX_DEFAULT

124472 The default mutex type. May be mapped to any of the above mutex types or may be
 124473 an implementation-defined type.

124474 *Normal* mutexes do not detect deadlock conditions; for example, a thread will hang if it
 124475 tries to relock a normal mutex that it already owns. Attempting to unlock a mutex locked
 124476 by another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal
 124477 mutexes will usually be the fastest type of mutex available on a platform but provide the
 124478 least error checking.

124479 *Recursive* mutexes are useful for converting old code where it is difficult to establish clear
 124480 boundaries of synchronization. A thread can relock a recursive mutex without first
 124481 unlocking it. The relocking deadlock which can occur with normal mutexes cannot occur
 124482 with this type of mutex. However, multiple locks of a recursive mutex require the same
 124483 number of unlocks to release the mutex before another thread can acquire the mutex.
 124484 Furthermore, this type of mutex maintains the concept of an owner. Thus, a thread
 124485 attempting to unlock a recursive mutex which another thread has locked returns with an
 124486 error. A thread attempting to unlock a recursive mutex that is not locked returns with an
 124487 error. Never use a recursive mutex with condition variables because the implicit unlock
 124488 performed by `pthread_cond_wait()` or `pthread_cond_timedwait()` will not actually release the
 124489 mutex if it had been locked multiple times.

124490 *Errorcheck* mutexes provide error checking and are useful primarily as a debugging aid. A
 124491 thread attempting to relock an errorcheck mutex without first unlocking it returns with an
 124492 error. Again, this type of mutex maintains the concept of an owner. Thus, a thread
 124493 attempting to unlock an errorcheck mutex which another thread has locked returns with
 124494 an error. A thread attempting to unlock an errorcheck mutex that is not locked also returns
 124495 with an error. It should be noted that errorcheck mutexes will almost always be much
 124496 slower than normal mutexes due to the extra state checks performed.

124497 The default mutex type provides implementation-defined error checking. The default
 124498 mutex may be mapped to one of the other defined types or may be something entirely
 124499 different. This enables each vendor to provide the mutex semantics which the vendor feels
 124500 will be most useful to their target users. Most vendors will probably choose to make
 124501 normal mutexes the default so as to give applications the benefit of the fastest type of

- 124502 mutexes available on their platform. Check your implementation's documentation.
- 124503 An application developer can use any of the mutex types almost interchangeably as long as the application does not depend upon the implementation detecting (or failing to
124504 detect) any particular errors. Note that a recursive mutex can be used with condition
124505 variable waits as long as the application never recursively locks the mutex.
124506
- 124507 Two functions are provided for manipulating the *type* attribute of a mutex attributes object.
124508 This attribute is set or returned in the *type* parameter of these functions. The
124509 `pthread_mutexattr_settype()` function is used to set a specific type value while
124510 `pthread_mutexattr_gettype()` is used to return the type of the mutex. Setting the *type*
124511 attribute of a mutex attributes object affects only mutexes initialized using that mutex
124512 attributes object. Changing the *type* attribute does not affect mutexes previously initialized
124513 using that mutex attributes object.
- 124514 Read-Write Locks and Attributes
- 124515 The read-write locks introduced have been harmonized with those in IEEE Std
124516 1003.1j-2000; see also [Section B.2.9.6](#) (on page 3659).
- 124517 Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock
124518 some shared data while updating that data, or allow any number of threads to have
124519 simultaneous read-only access to the data.
- 124520 Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A
124521 mutex excludes all other threads. A read-write lock allows other threads access to the data,
124522 providing no thread is modifying the data. Thus, a read-write lock is less primitive than
124523 either a mutex-condition variable pair or a semaphore.
- 124524 Application developers should consider using a read-write lock rather than a mutex to
124525 protect data that is frequently referenced but seldom modified. Most threads (readers) will
124526 be able to read the data without waiting and will only have to block when some other
124527 thread (a writer) is in the process of modifying the data. Conversely a thread that wants to
124528 change the data is forced to wait until there are no readers. This type of lock is often used
124529 to facilitate parallel access to data on multi-processor platforms or to avoid context
124530 switches on single processor platforms where multiple threads access the same data.
- 124531 If a read-write lock becomes unlocked and there are multiple threads waiting to acquire
124532 the write lock, the implementation's scheduling policy determines which thread acquires
124533 the read-write lock for writing. If there are multiple threads blocked on a read-write lock
124534 for both read locks and write locks, it is unspecified whether the readers or a writer
124535 acquire the lock first. However, for performance reasons, implementations often favor
124536 writers over readers to avoid potential writer starvation.
- 124537 A read-write lock object is an implementation-defined opaque object of type
124538 **pthread_rwlock_t** as defined in `<pthread.h>`. There are two different sorts of locks
124539 associated with a read-write lock: a read lock and a write lock.
- 124540 The `pthread_rwlockattr_init()` function initializes a read-write lock attributes object with the
124541 default value for all the attributes defined in the implementation. After a read-write lock
124542 attributes object has been used to initialize one or more read-write locks, changes to the
124543 read-write lock attributes object, including destruction, do not affect previously initialized
124544 read-write locks.
- 124545 Implementations must provide at least the read-write lock attribute *process-shared*. This
124546 attribute can have the following values:

124547	PTHREAD_PROCESS_SHARED
124548	Any thread of any process that has access to the memory where the read-write lock
124549	resides can manipulate the read-write lock.
124550	PTHREAD_PROCESS_PRIVATE
124551	Only threads created within the same process as the thread that initialized the read-
124552	write lock can manipulate the read-write lock. This is the default value.
124553	The <i>pthread_rwlockattr_setpshared()</i> function is used to set the <i>process-shared</i> attribute of an
124554	initialized read-write lock attributes object while the function
124555	<i>pthread_rwlockattr_getpshared()</i> obtains the current value of the <i>process-shared</i> attribute.
124556	A read-write lock attributes object is destroyed using the <i>pthread_rwlockattr_destroy()</i>
124557	function. The effect of subsequent use of the read-write lock attributes object is undefined.
124558	A thread creates a read-write lock using the <i>pthread_rwlock_init()</i> function. The attributes
124559	of the read-write lock can be specified by the application developer; otherwise, the default
124560	implementation-defined read-write lock attributes are used if the pointer to the read-write
124561	lock attributes object is NULL. In cases where the default attributes are appropriate, the
124562	PTHREAD_RWLOCK_INITIALIZER macro can be used to initialize read-write locks.
124563	A thread which wants to apply a read lock to the read-write lock can use either
124564	<i>pthread_rwlock_rdlock()</i> or <i>pthread_rwlock_tryrdlock()</i> . If <i>pthread_rwlock_rdlock()</i> is used, the
124565	thread acquires a read lock if a writer does not hold the write lock and there are no writers
124566	blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it
124567	can acquire a lock. However, if <i>pthread_rwlock_tryrdlock()</i> is used, the function returns
124568	immediately with the error [EBUSY] if any thread holds a write lock or there are blocked
124569	writers waiting for the write lock.
124570	A thread which wants to apply a write lock to the read-write lock can use either of two
124571	functions: <i>pthread_rwlock_wrlock()</i> or <i>pthread_rwlock_trywrlock()</i> . If <i>pthread_rwlock_wrlock()</i>
124572	is used, the thread acquires the write lock if no other reader or writer threads hold the
124573	read-write lock. If the write lock is not acquired, the thread blocks until it can acquire the
124574	write lock. However, if <i>pthread_rwlock_trywrlock()</i> is used, the function returns
124575	immediately with the error [EBUSY] if any thread is holding either a read or a write lock.
124576	The <i>pthread_rwlock_unlock()</i> function is used to unlock a read-write lock object held by the
124577	calling thread. Results are undefined if the read-write lock is not held by the calling thread.
124578	If there are other read locks currently held on the read-write lock object, the read-write
124579	lock object remains in the read locked state but without the current thread as one of its
124580	owners. If this function releases the last read lock for this read-write lock object, the read-
124581	write lock object is put in the unlocked read state. If this function is called to release a write
124582	lock for this read-write lock object, the read-write lock object is put in the unlocked state.
124583	Thread Concurrency Level
124584	On threads implementations that multiplex user threads onto a smaller set of kernel
124585	execution entities, the system attempts to create a reasonable number of kernel execution
124586	entities for the application upon application startup.
124587	On some implementations, these kernel entities are retained by user threads that block in
124588	the kernel. Other implementations do not <i>timeslice</i> user threads so that multiple compute-
124589	bound user threads can share a kernel thread. On such implementations, some
124590	applications may use up all the available kernel execution entities before their user-space
124591	threads are used up. The process may be left with user threads capable of doing work for
124592	the application but with no way to schedule them.
124593	The <i>pthread_setconcurrency()</i> function enables an application to request more kernel

124594 entities; that is, specify a desired concurrency level. However, this function merely
124595 provides a hint to the implementation. The implementation is free to ignore this request or
124596 to provide some other number of kernel entities. If an implementation does not multiplex
124597 user threads onto a smaller number of kernel execution entities, the
124598 *pthread_setconcurrency()* function has no effect.

124599 The *pthread_setconcurrency()* function may also have an effect on implementations where
124600 the kernel mode and user mode schedulers cooperate to ensure that ready user threads are
124601 not prevented from running by other threads blocked in the kernel.

124602 The *pthread_getconcurrency()* function always returns the value set by a previous call to
124603 *pthread_setconcurrency()*. However, if *pthread_setconcurrency()* was not previously called,
124604 this function returns zero to indicate that the threads implementation is maintaining the
124605 concurrency level.

124606 Thread Stack Guard Size

124607 DCE threads introduced the concept of a “thread stack guard size”. Most thread
124608 implementations add a region of protected memory to a thread’s stack, commonly known
124609 as a “guard region”, as a safety measure to prevent stack pointer overflow in one thread
124610 from corrupting the contents of another thread’s stack. The default size of the guard
124611 regions attribute is {PAGESIZE} bytes and is implementation-defined.

124612 Some application developers may wish to change the stack guard size. When an
124613 application creates a large number of threads, the extra page allocated for each stack may
124614 strain system resources. In addition to the extra page of memory, the kernel’s memory
124615 manager has to keep track of the different protections on adjoining pages. When this is a
124616 problem, the application developer may request a guard size of 0 bytes to conserve system
124617 resources by eliminating stack overflow protection.

124618 Conversely an application that allocates large data structures such as arrays on the stack
124619 may wish to increase the default guard size in order to detect stack overflow. If a thread
124620 allocates two pages for a data array, a single guard page provides little protection against
124621 thread stack overflows since the thread can corrupt adjoining memory beyond the guard
124622 page.

124623 The System Interfaces volume of POSIX.1-2017 defines a new attribute of a thread
124624 attributes object; that is, the *guardsize* attribute which allows applications to specify the size
124625 of the guard region of a thread’s stack.

124626 Two functions are provided for manipulating a thread’s stack guard size. The
124627 *pthread_attr_setguardsize()* function sets the thread *guardsize* attribute, and the
124628 *pthread_attr_getguardsize()* function retrieves the current value.

124629 An implementation may round up the requested guard size to a multiple of the
124630 configurable system variable {PAGESIZE}. In this case, *pthread_attr_getguardsize()* returns
124631 the guard size specified by the previous *pthread_attr_setguardsize()* function call and not
124632 the rounded up value.

124633 If an application is managing its own thread stacks using the *stackaddr* attribute, the
124634 *guardsize* attribute is ignored and no stack overflow protection is provided. In this case, it is
124635 the responsibility of the application to manage stack overflow along with stack allocation.

124636 Parallel I/O

124637 Suppose two or more threads independently issue read requests on the same file. To read
124638 specific data from a file, a thread must first call *lseek()* to seek to the proper offset in the
124639 file, and then call *read()* to retrieve the required data. If more than one thread does this at
124640 the same time, the first thread may complete its seek call, but before it gets a chance to

124641 issue its read call a second thread may complete its seek call, resulting in the first thread
 124642 accessing incorrect data when it issues its read call. One workaround is to lock the file
 124643 descriptor while seeking and reading or writing, but this reduces parallelism and adds
 124644 overhead.

124645 Instead, the System Interfaces volume of POSIX.1-2017 provides two functions to make
 124646 seek/read and seek/write operations atomic. The file descriptor's current offset is
 124647 unchanged, thus allowing multiple read and write operations to proceed in parallel. This
 124648 improves the I/O performance of threaded applications. The *pread()* function is used to do
 124649 an atomic read of data from a file into a buffer. Conversely, the *pwrite()* function does an
 124650 atomic write of data from a buffer to a file.

124651 B.2.9.1 Thread-Safety

124652 All functions required by POSIX.1-2017 need to be thread-safe. Implementations have to
 124653 provide internal synchronization when necessary in order to achieve this goal. In certain cases †
 124654 for example, most floating-point implementations †context switch code may have to manage
 124655 the writable shared state.

124656 While a read from a pipe of {PIPE_BUF}*2 bytes may not generate a single atomic and thread-
 124657 safe stream of bytes, it should generate "several" (individually atomic) thread-safe streams of
 124658 bytes. Similarly, while reading from a terminal device may not generate a single atomic and
 124659 thread-safe stream of bytes, it should generate some finite number of (individually atomic) and
 124660 thread-safe streams of bytes. That is, concurrent calls to read for a pipe, FIFO, or terminal device
 124661 are not allowed to result in corrupting the stream of bytes or other internal data. However,
 124662 *read()*, in these cases, is not required to return a single contiguous and atomic stream of bytes.

124663 It is not required that all functions provided by POSIX.1-2017 be either async-cancel-safe or
 124664 async-signal-safe.

124665 As it turns out, some functions are inherently not thread-safe; that is, their interface
 124666 specifications preclude async-signal-safety. For example, some functions (such as *asctime()*)
 124667 return a pointer to a result stored in memory space allocated by the function on a per-process
 124668 basis. Such a function is not thread-safe, because its result can be overwritten by successive
 124669 invocations. Other functions, while not inherently non-thread-safe, may be implemented in
 124670 ways that lead to them not being thread-safe. For example, some functions (such as *rand()*) store
 124671 state information (such as a seed value, which survives multiple function invocations) in
 124672 memory space allocated by the function on a per-process basis. The implementation of such a
 124673 function is not thread-safe if the implementation fails to synchronize invocations of the function
 124674 and thus fails to protect the state information. The problem is that when the state information is
 124675 not protected, concurrent invocations can interfere with one another (for example, applications
 124676 using *rand()* may see the same seed value).

124677 Thread-Safety and Locking of Existing Functions

124678 Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some
 124679 implementations of some existing functions will not work properly when executed concurrently.
 124680 To provide routines that will work correctly in an environment with threads ("thread-safe"), two
 124681 problems need to be solved:

- 124682 1. Routines that maintain or return pointers to static areas internal to the routine (which
 124683 may now be shared) need to be modified. The routines *ttyname()* and *localtime()* are
 124684 examples.
- 124685 2. Routines that access data space shared by more than one thread need to be modified. The
 124686 *malloc()* function and the *stdio* family routines are examples.

124687 There are a variety of constraints on these changes. The first is compatibility with the existing
 124688 versions of these functions—non-thread-safe functions will continue to be in use for some time,
 124689 as the original interfaces are used by existing code. Another is that the new thread-safe versions
 124690 of these functions represent as small a change as possible over the familiar interfaces provided
 124691 by the existing non-thread-safe versions. The new interfaces should be independent of any
 124692 particular threads implementation. In particular, they should be thread-safe without depending
 124693 on explicit thread-specific memory. Finally, there should be minimal performance penalty due to
 124694 the changes made to the functions.

124695 It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for
 124696 which corrected versions are provided be complete.

124697 *Thread-Safety and Locking Solutions*

124698 Many of the POSIX.1 functions were thread-safe and did not change at all. However, some
 124699 functions (for example, the math functions typically found in **libm**) are not thread-safe because
 124700 of writable shared global state. For instance, in IEEE Std 754-1985 floating-point
 124701 implementations, the computation modes and flags are global and shared.

124702 Some functions are not thread-safe because a particular implementation is not reentrant,
 124703 typically because of a non-essential use of static storage. These require only a new
 124704 implementation.

124705 Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming
 124706 environments, not just within pthreads. In order to be used outside the context of pthreads,
 124707 however, such libraries still have to use some synchronization method. These could either be
 124708 independent of the pthread synchronization operations, or they could be a subset of the pthread
 124709 interfaces. Either method results in thread-safe library implementations that can be used without
 124710 the rest of pthreads.

124711 Some functions, such as the *stdio* family interface and dynamic memory allocation functions
 124712 such as *malloc()*, are inter-dependent routines that share resources (for example, buffers) across
 124713 related calls. These require synchronization to work correctly, but they do not require any
 124714 change to their external (user-visible) interfaces.

124715 In some cases, such as *getc()* and *putc()*, adding synchronization is likely to create an
 124716 unacceptable performance impact. In this case, slower thread-safe synchronized functions are to
 124717 be provided, but the original, faster (but unsafe) functions (which may be implemented as
 124718 macros) are retained under new names. Some additional special-purpose synchronization
 124719 facilities are necessary for these macros to be usable in multi-threaded programs. This also
 124720 requires changes in **<stdio.h>**.

124721 The other common reason that functions are unsafe is that they return a pointer to static storage,
 124722 making the functions non-thread-safe. This has to be changed, and there are three natural
 124723 choices:

124724 1. Return a pointer to thread-specific storage

124725 This could incur a severe performance penalty on those architectures with a costly
 124726 implementation of the thread-specific data interface.

124727 A variation on this technique is to use *malloc()* to allocate storage for the function output
 124728 and return a pointer to this storage. This technique may also have an undesirable
 124729 performance impact, however, and a simplistic implementation requires that the user
 124730 program explicitly free the storage object when it is no longer needed. This technique is
 124731 used by some existing POSIX.1 functions. With careful implementation for infrequently
 124732 used functions, there may be little or no performance or storage penalty, and the
 124733 maintenance of already-standardized interfaces is a significant benefit.

- 124734 2. Return the actual value computed by the function
- 124735 This technique can only be used with functions that return pointers to structures—
 124736 routines that return character strings would have to wrap their output in an enclosing
 124737 structure in order to return the output on the stack. There is also a negative performance
 124738 impact inherent in this solution in that the output value has to be copied twice before it
 124739 can be used by the calling function: once from the called routine’s local buffers to the top
 124740 of the stack, then from the top of the stack to the assignment target. Finally, many older
 124741 compilers cannot support this technique due to a historical tendency to use internal static
 124742 buffers to deliver the results of structure-valued functions.
- 124743 3. Have the caller pass the address of a buffer to contain the computed value
- 124744 The only disadvantage of this approach is that extra arguments have to be provided by
 124745 the calling program. It represents the most efficient solution to the problem, however,
 124746 and, unlike the `malloc()` technique, it is semantically clear.
- 124747 There are some routines (often groups of related routines) whose interfaces are inherently non-
 124748 thread-safe because they communicate across multiple function invocations by means of static
 124749 memory locations. The solution is to redesign the calls so that they are thread-safe, typically by
 124750 passing the needed data as extra parameters. Unfortunately, this may require major changes to
 124751 the interface as well.
- 124752 A floating-point implementation using IEEE Std 754-1985 is a case in point. A less problematic
 124753 example is the `rand48` family of pseudo-random number generators. The functions `getgrgid()`,
 124754 `getgrnam()`, `getpwnam()`, and `getpwuid()` are another such case.
- 124755 The problems with `errno` are discussed in [Alternative Solutions for Per-Thread `errno`](#) (on page
 124756 3574).
- 124757 Some functions can be thread-safe or not, depending on their arguments. These include the
 124758 `tmpnam()` and `ctermid()` functions. These functions have pointers to character strings as
 124759 arguments. If the pointers are not NULL, the functions store their results in the character string;
 124760 however, if the pointers are NULL, the functions store their results in an area that may be static
 124761 and thus subject to overwriting by successive calls. These should only be called by multi-thread
 124762 applications when their arguments are non-NULL.
- 124763 *Asynchronous Safety and Thread-Safety*
- 124764 A floating-point implementation has many modes that effect rounding and other aspects of
 124765 computation. Functions in some math library implementations may change the computation
 124766 modes for the duration of a function call. If such a function call is interrupted by a signal or
 124767 cancellation, the floating-point state is not required to be protected.
- 124768 There is a significant cost to make floating-point operations async-cancel-safe or async-signal-
 124769 safe; accordingly, neither form of async safety is required.
- 124770 *Functions Returning Pointers to Static Storage*
- 124771 For those functions that are not thread-safe because they return values in fixed size statically
 124772 allocated structures, alternate “_r” forms are provided that pass a pointer to an explicit result
 124773 structure. Those that return pointers into library-allocated buffers have forms provided with
 124774 explicit buffer and length parameters.
- 124775 For functions that return pointers to library-allocated buffers, it makes sense to provide “_r”
 124776 versions that allow the application control over allocation of the storage in which results are
 124777 returned. This allows the state used by these functions to be managed on an application-specific
 124778 basis, supporting per-thread, per-process, or other application-specific sharing relationships.
- 124779 Early proposals had provided “_r” versions for functions that returned pointers to variable-size

124780 buffers without providing a means for determining the required buffer size. This would have
 124781 made using such functions exceedingly clumsy, potentially requiring iteratively calling them
 124782 with increasingly larger guesses for the amount of storage required. Hence, *sysconf()* variables
 124783 have been provided for such functions that return the maximum required buffer size.

124784 Thus, the rule that has been followed by POSIX.1-2017 when adapting single-threaded non-
 124785 thread-safe functions is as follows: all functions returning pointers to library-allocated storage
 124786 should have ```_r``` versions provided, allowing the application control over the storage
 124787 allocation. Those with variable-sized return values accept both a buffer address and a length
 124788 parameter. The *sysconf()* variables are provided to supply the appropriate buffer sizes when
 124789 required. Implementors are encouraged to apply the same rule when adapting their own
 124790 existing functions to a pthreads environment.

124791 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0020 [631], XSH/TC2-2008/0021 [826],
 124792 and XSH/TC2-2008/0022 [631] are applied.

124793 B.2.9.2 Thread IDs

124794 Separate applications should communicate through well-defined interfaces and should not
 124795 depend on each other's implementation. For example, if a programmer decides to rewrite the
 124796 *sort* utility using multiple threads, it should be easy to do this so that the interface to the *sort*
 124797 utility does not change. Consider that if the user causes SIGINT to be generated while the *sort*
 124798 utility is running, keeping the same interface means that the entire *sort* utility is killed, not just
 124799 one of its threads. As another example, consider a realtime application that manages a reactor.
 124800 Such an application may wish to allow other applications to control the priority at which it
 124801 watches the control rods. One technique to accomplish this is to write the ID of the thread
 124802 watching the control rods into a file and allow other programs to change the priority of that
 124803 thread as they see fit. A simpler technique is to have the reactor process accept IPCs
 124804 (Interprocess Communication messages) from other processes, telling it at a semantic level what
 124805 priority the program should assign to watching the control rods. This allows the programmer
 124806 greater flexibility in the implementation. For example, the programmer can change the
 124807 implementation from having one thread per rod to having one thread watching all of the rods
 124808 without changing the interface. Having threads live inside the process means that the
 124809 implementation of a process is invisible to outside processes (excepting debuggers and system
 124810 management tools).

124811 Threads do not provide a protection boundary. Every thread model allows threads to share
 124812 memory with other threads and encourages this sharing to be widespread. This means that one
 124813 thread can wipe out memory that is needed for the correct functioning of other threads that are
 124814 sharing its memory. Consequently, providing each thread with its own user and/or group IDs
 124815 would not provide a protection boundary between threads sharing memory.

124816 Some applications make the assumption that the implementation can always detect invalid uses
 124817 of thread IDs of type `pthread_t`. This is an invalid assumption. Specifically, if `pthread_t`
 124818 is defined as a pointer type, no access check needs to be performed before using the ID.

124819 As with other interfaces that take pointer parameters, the outcome of passing an invalid
 124820 parameter can result in an invalid memory reference or an attempt to access an undefined
 124821 portion of a memory object, cause signals to be sent (SIGSEGV or SIGBUS) and possible
 124822 termination of the process. This is a similar case to passing an invalid buffer pointer to *read()*.
 124823 Some implementations might implement *read()* as a system call and set an [EFAULT] error
 124824 condition. Other implementations might contain parts of *read()* at user level and the first
 124825 attempt to access data at an invalid reference will cause a signal to be sent instead.

124826 If an implementation detects use of a thread ID after the end of its lifetime, it is recommended
 124827 that the function should fail and report an [ESRCH] error. This does not imply that

124828 implementations are required to return in this case. It is legitimate behavior to send an “invalid
 124829 memory reference” signal (SIGSEGV or SIGBUS). It is the application’s responsibility to use only
 124830 valid thread IDs and to keep track of the lifetime of the underlying threads.

124831 *B.2.9.3 Thread Mutexes*

124832 There is no additional rationale provided for this section.

124833 *B.2.9.4 Thread Scheduling*

124834 Scheduling Implementation Models

124835 The following scheduling implementation models are presented in terms of threads and
 124836 “kernel entities”. This is to simplify exposition of the models, and it does not imply that
 124837 an implementation actually has an identifiable “kernel entity”.

124838 A kernel entity is not defined beyond the fact that it has scheduling attributes that are used
 124839 to resolve contention with other kernel entities for execution resources. A kernel entity
 124840 may be thought of as an envelope that holds a thread or a separate kernel thread. It is not a
 124841 conventional process, although it shares with the process the attribute that it has a single
 124842 thread of control; it does not necessarily imply an address space, open files, and so on. It is
 124843 better thought of as a primitive facility upon which conventional processes and threads
 124844 may be constructed.

124845 *System Thread Scheduling Model*

124846 This model consists of one thread per kernel entity. The kernel entity is solely
 124847 responsible for scheduling thread execution on one or more processors. This model
 124848 schedules all threads against all other threads in the system using the scheduling
 124849 attributes of the thread.

124850 *Process Scheduling Model*

124851 A generalized process scheduling model consists of two levels of scheduling. A
 124852 threads library creates a pool of kernel entities, as required, and schedules threads to
 124853 run on them using the scheduling attributes of the threads. Typically, the size of the
 124854 pool is a function of the simultaneously runnable threads, not the total number of
 124855 threads. The kernel then schedules the kernel entities onto processors according to
 124856 their scheduling attributes, which are managed by the threads library. This set model
 124857 potentially allows a wide range of mappings between threads and kernel entities.

124858 *System and Process Scheduling Model Performance*

124859 There are a number of important implications on the performance of applications using
 124860 these scheduling models. The process scheduling model potentially provides lower
 124861 overhead for making scheduling decisions, since there is no need to access kernel-level
 124862 information or functions and the set of schedulable entities is smaller (only the threads
 124863 within the process).

124864 On the other hand, since the kernel is also making scheduling decisions regarding the
 124865 system resources under its control (for example, CPU(s), I/O devices, memory), decisions
 124866 that do not take thread scheduling parameters into account can result in unspecified
 124867 delays for realtime application threads, causing them to miss maximum response time
 124868 limits.

- 124869 Rate Monotonic Scheduling
- 124870 Rate monotonic scheduling was considered, but rejected for standardization in the context
124871 of pthreads. A sporadic server policy is included.
- 124872 Scheduling Options
- 124873 In POSIX.1-2017, the basic thread scheduling functions are defined under the threads
124874 functionality, so that they are required of all threads implementations. However, there are
124875 no specific scheduling policies required by this functionality to allow for conforming
124876 thread implementations that are not targeted to realtime applications.
- 124877 Specific standard scheduling policies are defined to be under the Thread Execution
124878 Scheduling option, and they are specifically designed to support realtime applications by
124879 providing predictable resource-sharing sequences. The name of this option was chosen to
124880 emphasize that this functionality is defined as appropriate for realtime applications that
124881 require simple priority-based scheduling.
- 124882 It is recognized that these policies are not necessarily satisfactory for some multi-processor
124883 implementations, and work is ongoing to address a wider range of scheduling behaviors.
124884 The interfaces have been chosen to create abundant opportunity for future scheduling
124885 policies to be implemented and standardized based on this interface. In order to
124886 standardize a new scheduling policy, all that is required (from the standpoint of thread
124887 scheduling attributes) is to define a new policy name, new members of the thread
124888 attributes object, and functions to set these members when the scheduling policy is equal
124889 to the new value.
- 124890 **Scheduling Contention Scope**
- 124891 In order to accommodate the requirement for realtime response, each thread has a scheduling
124892 contention scope attribute. Threads with a system scheduling contention scope have to be
124893 scheduled with respect to all other threads in the system. These threads are usually bound to a
124894 single kernel entity that reflects their scheduling attributes and are directly scheduled by the
124895 kernel.
- 124896 Threads with a process scheduling contention scope need be scheduled only with respect to the
124897 other threads in the process. These threads may be scheduled within the process onto a pool of
124898 kernel entities. The implementation is also free to bind these threads directly to kernel entities
124899 and let them be scheduled by the kernel. Process scheduling contention scope allows the
124900 implementation the most flexibility and is the default if both contention scopes are supported
124901 and none is specified.
- 124902 Thus, the choice by implementors to provide one or the other (or both) of these scheduling
124903 models is driven by the need of their supported application domains for worst-case (that is,
124904 realtime) response, or average-case (non-realtime) response.
- 124905 **Scheduling Allocation Domain**
- 124906 The SCHED_FIFO and SCHED_RR scheduling policies take on different characteristics on a
124907 multi-processor. Other scheduling policies are also subject to changed behavior when executed
124908 on a multi-processor. The concept of scheduling allocation domain determines the set of
124909 processors on which the threads of an application may run. By considering the application's
124910 processor scheduling allocation domain for its threads, scheduling policies can be defined in
124911 terms of their behavior for varying processor scheduling allocation domain values. It is
124912 conceivable that not all scheduling allocation domain sizes make sense for all scheduling
124913 policies on all implementations. The concept of scheduling allocation domain, however, is a
124914 useful tool for the description of multi-processor scheduling policies.

124915 The “process control” approach to scheduling obtains significant performance advantages from
124916 dynamic scheduling allocation domain sizes when it is applicable.

124917 Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure
124918 that involves reassignment of threads among scheduling allocation domains. In NUMA
124919 machines, a natural model of scheduling is to match scheduling allocation domains to clusters of
124920 processors. Load balancing in such an environment requires changing the scheduling allocation
124921 domain to which a thread is assigned.

124922 **Scheduling Documentation**

124923 Implementation-provided scheduling policies need to be completely documented in order to be
124924 useful. This documentation includes a description of the attributes required for the policy, the
124925 scheduling interaction of threads running under this policy and all other supported policies, and
124926 the effects of all possible values for processor scheduling allocation domain. Note that for the
124927 implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the
124928 behavior as undefined.

124929 **Scheduling Contention Scope Attribute**

124930 The scheduling contention scope defines how threads compete for resources. Within
124931 POSIX.1-2017, scheduling contention scope is used to describe only how threads are scheduled
124932 in relation to one another in the system. That is, either they are scheduled against all other
124933 threads in the system (“system scope”) or only against those threads in the process (“process
124934 scope”). In fact, scheduling contention scope may apply to additional resources, including
124935 virtual timers and profiling, which are not currently considered by POSIX.1-2017.

124936 **Mixed Scopes**

124937 If only one scheduling contention scope is supported, the scheduling decision is straightforward.
124938 To perform the processor scheduling decision in a mixed scope environment, it is necessary to
124939 map the scheduling attributes of the thread with process-wide contention scope to the same
124940 attribute space as the thread with system-wide contention scope.

124941 Since a conforming implementation has to support one and may support both scopes, it is useful
124942 to discuss the effects of such choices with respect to example applications. If an implementation
124943 supports both scopes, mixing scopes provides a means of better managing system-level (that is,
124944 kernel-level) and library-level resources. In general, threads with system scope will require the
124945 resources of a separate kernel entity in order to guarantee the scheduling semantics. On the
124946 other hand, threads with process scope can share the resources of a kernel entity while
124947 maintaining the scheduling semantics.

124948 The application is free to create threads with dedicated kernel resources, and other threads that
124949 multiplex kernel resources. Consider the example of a window server. The server allocates two
124950 threads per widget: one thread manages the widget user interface (including drawing), while
124951 the other thread takes any required application action. This allows the widget to be “active”
124952 while the application is computing. A screen image may be built from thousands of widgets. If
124953 each of these threads had been created with system scope, then most of the kernel-level
124954 resources might be wasted, since only a few widgets are active at any one time. In addition,
124955 mixed scope is particularly useful in a window server where one thread with high priority and
124956 system scope handles the mouse so that it tracks well. As another example, consider a database
124957 server. For each of the hundreds or thousands of clients supported by a large server, an
124958 equivalent number of threads will have to be created. If each of these threads were system scope,
124959 the consequences would be the same as for the window server example above. However, the
124960 server could be constructed so that actual retrieval of data is done by several dedicated threads.
124961 Dedicated threads that do work for all clients frequently justify the added expense of system

124962 scope. If it were not permissible to mix system and process threads in the same process, this type
124963 of solution would not be possible.

124964 **Dynamic Thread Scheduling Parameters Access**

124965 In many time-constrained applications, there is no need to change the scheduling attributes
124966 dynamically during thread or process execution, since the general use of these attributes is to
124967 reflect directly the time constraints of the application. Since these time constraints are generally
124968 imposed to meet higher-level system requirements, such as accuracy or availability, they
124969 frequently should remain unchanged during application execution.

124970 However, there are important situations in which the scheduling attributes should be changed.
124971 Generally, this will occur when external environmental conditions exist in which the time
124972 constraints change. Consider, for example, a space vehicle major mode change, such as the
124973 change from ascent to descent mode, or the change from the space environment to the
124974 atmospheric environment. In such cases, the frequency with which many of the sensors or
124975 actuators need to be read or written will change, which will necessitate a priority change. In
124976 other cases, even the existence of a time constraint might be temporary, necessitating not just a
124977 priority change, but also a policy change for ongoing threads or processes. For this reason, it is
124978 critical that the interface should provide functions to change the scheduling parameters
124979 dynamically, but, as with many of the other realtime functions, it is important that applications
124980 use them properly to avoid the possibility of unnecessarily degrading performance.

124981 In providing functions for dynamically changing the scheduling behavior of threads, there were
124982 two options: provide functions to get and set the individual scheduling parameters of threads,
124983 or provide a single interface to get and set all the scheduling parameters for a given thread
124984 simultaneously. Both approaches have merit. Access functions for individual parameters allow
124985 simpler control of thread scheduling for simple thread scheduling parameters. However, a single
124986 function for setting all the parameters for a given scheduling policy is required when first setting
124987 that scheduling policy. Since the single all-encompassing functions are required, it was decided
124988 to leave the interface as minimal as possible. Note that simpler functions (such as
124989 *pthread_setprio()* for threads running under the priority-based schedulers) can be easily defined
124990 in terms of the all-encompassing functions.

124991 If the *pthread_setschedparam()* function executes successfully, it will have set all of the scheduling
124992 parameter values indicated in *param*; otherwise, none of the scheduling parameters will have
124993 been modified. This is necessary to ensure that the scheduling of this and all other threads
124994 continues to be consistent in the presence of an erroneous scheduling parameter.

124995 The [EPERM] error value is included in the list of possible *pthread_setschedparam()* error returns
124996 as a reflection of the fact that the ability to change scheduling parameters increases risks to the
124997 implementation and application performance if the scheduling parameters are changed
124998 improperly. For this reason, and based on some existing practice, it was felt that some
124999 implementations would probably choose to define specific permissions for changing either a
125000 thread's own or another thread's scheduling parameters. POSIX.1-2017 does not include
125001 portable methods for setting or retrieving permissions, so any such use of permissions is
125002 completely unspecified.

125003 Mutex Initialization Scheduling Attributes

125004 In a priority-driven environment, a direct use of traditional primitives like mutexes and
125005 condition variables can lead to unbounded priority inversion, where a higher priority thread can
125006 be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a
125007 result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded
125008 and minimized by the use of priority inheritance protocols. This allows thread deadlines to be
125009 guaranteed even in the presence of synchronization requirements.

125010 Two useful but simple members of the family of priority inheritance protocols are the basic
125011 priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic
125012 Priority Inheritance protocol (governed by the Non-Robust Mutex Priority Inheritance option), a
125013 thread that is blocking higher priority threads executes at the priority of the highest priority
125014 thread that it blocks. This simple mechanism allows priority inversion to be bounded by the
125015 duration of critical sections and makes timing analysis possible.

125016 Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority
125017 Protection option), each mutex has a priority ceiling, usually defined as the priority of the
125018 highest priority thread that can lock the mutex. When a thread is executing inside critical
125019 sections, its priority is unconditionally increased to the highest of the priority ceilings of all the
125020 mutexes owned by the thread. This protocol has two very desirable properties in uni-processor
125021 systems. First, a thread can be blocked by a lower priority thread for at most the duration of one
125022 single critical section. Furthermore, when the protocol is correctly used in a single processor, and
125023 if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

125024 The priority ceiling emulation can be extended to multiple processor environments, in which
125025 case the values of the priority ceilings will be assigned depending on the kind of mutex that is
125026 being used: local to only one processor, or global, shared by several processors. Local priority
125027 ceilings will be assigned the usual way, equal to the priority of the highest priority thread that
125028 may lock that mutex. Global priority ceilings will usually be assigned a priority level higher
125029 than all the priorities assigned to any of the threads that reside in the involved processors to
125030 avoid the effect called remote blocking.

125031 Change the Priority Ceiling of a Mutex

125032 In order for the priority protect protocol to exhibit its desired properties of bounding priority
125033 inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same
125034 as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the
125035 threads using such mutexes never change dynamically, there is no need ever to change the
125036 priority ceiling of a mutex.

125037 However, if a major system mode change results in an altered response time requirement for one
125038 or more application threads, their priority has to change to reflect it. It will occasionally be the
125039 case that the priority ceilings of mutexes held also need to change. While changing priority
125040 ceilings should generally be avoided, it is important that POSIX.1-2017 provide these interfaces
125041 for those cases in which it is necessary.

125042 B.2.9.5 Thread Cancellation

125043 Many existing threads packages have facilities for canceling an operation or canceling a thread.
125044 These facilities are used for implementing user requests (such as the CANCEL button in a
125045 window-based application), for implementing OR parallelism (for example, telling the other
125046 threads to stop working once one thread has found a forced mate in a parallel chess program), or
125047 for implementing the ABORT mechanism in Ada.

125048 POSIX programs traditionally have used the signal mechanism combined with either *longjmp()*

125049 or polling to cancel operations. Many POSIX programmers have trouble using these facilities to
125050 solve their problems efficiently in a single-threaded process. With the introduction of threads,
125051 these solutions become even more difficult to use.

125052 The main issues with implementing a cancellation facility are specifying the operation to be
125053 canceled, cleanly releasing any resources allocated to that operation, controlling when the target
125054 notices that it has been canceled, and defining the interaction between asynchronous signals and
125055 cancellation.

125056 **Specifying the Operation to Cancel**

125057 Consider a thread that calls through five distinct levels of program abstraction and then, inside
125058 the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary
125059 is a layer at which the client of the abstraction sees only the service being provided and can
125060 remain ignorant of the implementation. Abstractions are often layered, each level of abstraction
125061 being a client of the lower-level abstraction and implementing a higher-level abstraction.)
125062 Depending on the semantics of each abstraction, one could imagine wanting to cancel only the
125063 call that causes suspension, only the bottom two levels, or the operation being done by the entire
125064 thread. Canceling operations at a finer grain than the entire thread is difficult because threads
125065 are active and they may be run in parallel on a multi-processor. By the time one thread can make
125066 a request to cancel an operation, the thread performing the operation may have completed that
125067 operation and gone on to start another operation whose cancellation is not desired. Thread IDs
125068 are not reused until the thread has exited, and either it was created with the *Attr detachstate*
125069 attribute set to *PTHREAD_CREATE_DETACHED* or the *pthread_join()* or *pthread_detach()*
125070 function has been called for that thread. Consequently, a thread cancellation will never be
125071 misdirected when the thread terminates. For these reasons, the canceling of operations is done at
125072 the granularity of the thread. Threads are designed to be inexpensive enough so that a separate
125073 thread may be created to perform each separately cancelable operation; for example, each
125074 possibly long running user request.

125075 For cancellation to be used in existing code, cancellation scopes and handlers will have to be
125076 established for code that needs to release resources upon cancellation, so that it follows the
125077 programming discipline described in the text.

125078 **A Special Signal Versus a Special Interface**

125079 Two different mechanisms were considered for providing the cancellation interfaces. The first
125080 was to provide an interface to direct signals at a thread and then to define a special signal that
125081 had the required semantics. The other alternative was to use a special interface that delivered the
125082 correct semantics to the target thread.

125083 The solution using signals produced a number of problems. It required the implementation to
125084 provide cancellation in terms of signals whereas a perfectly valid (and possibly more efficient)
125085 implementation could have both layered on a low-level set of primitives. There were so many
125086 exceptions to the special signal (it cannot be used with *kill()*, no POSIX.1 interfaces can be used
125087 with it) that it was clearly not a valid signal. Its semantics on delivery were also completely
125088 different from any existing POSIX.1 signal. As such, a special interface that did not mandate the
125089 implementation and did not confuse the semantics of signals and cancellation was felt to be the
125090 better solution.

125091 Races Between Cancellation and Resuming Execution

125092 Due to the nature of cancellation, there is generally no synchronization between the thread
125093 requesting the cancellation of a blocked thread and events that may cause that thread to resume
125094 execution. For this reason, and because excess serialization hurts performance, when both an
125095 event that a thread is waiting for has occurred and a cancellation request has been made and
125096 cancellation is enabled, POSIX.1-2017 explicitly allows the implementation to choose between
125097 returning from the blocking call or acting on the cancellation request.

125098 Interaction of Cancellation with Asynchronous Signals

125099 A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation
125100 cleanup handler for releasing the resource when and if the thread is canceled.

125101 A correct and complete implementation of cancellation in the presence of asynchronous signals
125102 requires considerable care. An implementation has to push a cancellation cleanup handler on the
125103 cancellation cleanup stack while maintaining the integrity of the stack data structure. If an
125104 asynchronously-generated signal is posted to the thread during a stack operation, the signal
125105 handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous
125106 signal handlers may not cancel threads or otherwise manipulate the cancellation state of a
125107 thread. Threads may, of course, be canceled by another thread that used a *sigwait()* function to
125108 wait synchronously for an asynchronous signal.

125109 In order for cancellation to function correctly, it is required that asynchronous signal handlers
125110 not change the cancellation state. This requires that some elements of existing practice, such as
125111 using *longjmp()* to exit from an asynchronous signal handler implicitly, be prohibited in cases
125112 where the integrity of the cancellation state of the interrupt thread cannot be ensured.

125113 Thread Cancellation Overview**125114 Cancelability States**

125115 The three possible cancelability states (disabled, deferred, and asynchronous) are encoded
125116 into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be
125117 changed and restored independently. For instance, short code sequences that will not block
125118 sometimes disable cancelability on entry and restore the previous state upon exit.
125119 Likewise, long or unbounded code sequences containing no convenient explicit
125120 cancellation points will sometimes set the cancelability type to asynchronous on entry and
125121 restore the previous value upon exit.

125122 Cancellation Points

125123 Cancellation points are points inside of certain functions where a thread has to act on any
125124 pending cancellation request when cancelability is enabled. For functions in the “shall
125125 occur” list, a cancellation check must be performed on every call regardless of whether,
125126 absent the cancellation, the call would have blocked. For functions in the “may occur” list,
125127 a cancellation check may be performed on some calls but not others; i.e., whether or not a
125128 cancellation point occurs when one of these functions is being executed can depend on
125129 current conditions.

125130 The idea was considered of allowing implementations to define whether blocking calls
125131 such as *read()* should be cancellation points. It was decided that it would adversely affect
125132 the design of conforming applications if blocking calls were not cancellation points
125133 because threads could be left blocked in an uncancelable state.

125134 There are several important blocking routines that are specifically not made cancellation
125135 points:

125136 ‡*pthread_mutex_lock()*

125137 If *pthread_mutex_lock()* were a cancellation point, every routine that called it would
 125138 also become a cancellation point (that is, any routine that touched shared state would
 125139 automatically become a cancellation point). For example, *malloc()*, *free()*, and *rand()*
 125140 would become cancellation points under this scheme. Having too many cancellation
 125141 points makes programming very difficult, leading to either much disabling and
 125142 restoring of cancelability or much difficulty in trying to arrange for reliable cleanup
 125143 at every possible place.

125144 Since *pthread_mutex_lock()* is not a cancellation point, threads could result in being
 125145 blocked uninterruptibly for long periods of time if mutexes were used as a general
 125146 synchronization mechanism. As this is normally not acceptable, mutexes should only
 125147 be used to protect resources that are held for small fixed lengths of time where not
 125148 being able to be canceled will not be a problem. Resources that need to be held
 125149 exclusively for long periods of time should be protected with condition variables.

125150 ‡*pthread_barrier_wait()*

125151 Canceling a barrier wait will render a barrier unusable. Similar to a barrier timeout
 125152 (which the standard developers rejected), there is no way to guarantee the
 125153 consistency of a barrier's internal data structures if a barrier wait is canceled.

125154 ‡*pthread_spin_lock()*

125155 As with mutexes, spin locks should only be used to protect resources that are held for
 125156 small fixed lengths of time where not being cancelable will not be a problem.

125157 Every library routine should specify whether or not it includes any cancellation points.
 125158 Typically, only those routines that may block or compute indefinitely need to include
 125159 cancellation points.

125160 Correctly coded routines only reach cancellation points after having set up a cancellation
 125161 cleanup handler to restore invariants if the thread is canceled at that point. Being
 125162 cancelable only at specified cancellation points allows programmers to keep track of
 125163 actions needed in a cancellation cleanup handler more easily. A thread should only be
 125164 made asynchronously cancelable when it is not in the process of acquiring or releasing
 125165 resources or otherwise in a state from which it would be difficult or impossible to recover.

125166 Thread Cancellation Cleanup Handlers

125167 The cancellation cleanup handlers provide a portable mechanism, easy to implement, for
 125168 releasing resources and restoring invariants. They are easier to use than signal handlers
 125169 because they provide a stack of cancellation cleanup handlers rather than a single handler,
 125170 and because they have an argument that can be used to pass context information to the
 125171 handler.

125172 The alternative to providing these simple cancellation cleanup handlers (whose only use is
 125173 for cleaning up when a thread is canceled) is to define a general exception package that
 125174 could be used for handling and cleaning up after hardware traps and software-detected
 125175 errors. This was too far removed from the charter of providing threads to handle
 125176 asynchrony. However, it is an explicit goal of POSIX.1-2017 to be compatible with existing
 125177 exception facilities and languages having exceptions.

125178 The interaction of this facility and other procedure-based or language-level exception
 125179 facilities is unspecified in this version of POSIX.1-2017. However, it is intended that it be
 125180 possible for an implementation to define the relationship between these cancellation
 125181 cleanup handlers and Ada, C++, or other language-level exception handling facilities.

125182 It was suggested that the cancellation cleanup handlers should also be called when the
 125183 process exits or calls the *exec* function. This was rejected partly due to the performance
 125184 problem caused by having to call the cancellation cleanup handlers of every thread before
 125185 the operation could continue. The other reason was that the only state expected to be
 125186 cleaned up by the cancellation cleanup handlers would be the intraprocess state. Any
 125187 handlers that are to clean up the interprocess state would be registered with *atexit()*. There
 125188 is the orthogonal problem that the *exec* functions do not honor the *atexit()* handlers, but
 125189 resolving this is beyond the scope of POSIX.1-2017.

125190 Async-Cancel Safety

125191 A function is said to be async-cancel-safe if it is written in such a way that entering the
 125192 function with asynchronous cancelability enabled will not cause any invariants to be
 125193 violated, even if a cancellation request is delivered at any arbitrary instruction. Functions
 125194 that are async-cancel-safe are often written in such a way that they need to acquire no
 125195 resources for their operation and the visible variables that they may write are strictly
 125196 limited.

125197 Any routine that gets a resource as a side-effect cannot be made async-cancel-safe (for
 125198 example, *malloc()*). If such a routine were called with asynchronous cancelability enabled,
 125199 it might acquire the resource successfully, but as it was returning to the client, it could act
 125200 on a cancellation request. In such a case, the application would have no way of knowing
 125201 whether the resource was acquired or not.

125202 Indeed, because many interesting routines cannot be made async-cancel-safe, most library
 125203 routines in general are not async-cancel-safe. Every library routine should specify whether
 125204 or not it is async-cancel safe so that programmers know which routines can be called from
 125205 code that is asynchronously cancelable.

125206 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/8 is applied, adding the *pselect()* function
 125207 to the list of functions with cancellation points.

125208 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/5 is applied, adding the *fdatasync()*
 125209 function into the table of functions that shall have cancellation points.

125210 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/6 is applied, adding the numerous
 125211 functions into the table of functions that may have cancellation points.

125212 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/7 is applied, clarifying the requirements
 125213 in Thread Cancellation Cleanup Handlers.

125214 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0023 [627], XSH/TC2-2008/0024
 125215 [627,632], XSH/TC2-2008/0025 [627], XSH/TC2-2008/0026 [632], and XSH/TC2-2008/0027
 125216 [622] are applied.

125217 B.2.9.6 *Thread Read-Write Locks*

125218 **Background**

125219 Read-write locks are often used to allow parallel access to data on multi-processors, to avoid
 125220 context switches on uni-processors when multiple threads access the same data, and to protect
 125221 data structures that are frequently accessed (that is, read) but rarely updated (that is, written).
 125222 The in-core representation of a file system directory is a good example of such a data structure.
 125223 One would like to achieve as much concurrency as possible when searching directories, but limit
 125224 concurrent access when adding or deleting files.

125225 Although read-write locks can be implemented with mutexes and condition variables, such
 125226 implementations are significantly less efficient than is possible. Therefore, this synchronization

125227 primitive is included in POSIX.1-2017 for the purpose of allowing more efficient
125228 implementations in multi-processor systems.

125229 **Queuing of Waiting Threads**

125230 The `pthread_rwlock_unlock()` function description states that one writer or one or more readers
125231 must acquire the lock if it is no longer held by any thread as a result of the call. However, the
125232 function does not specify which thread(s) acquire the lock, unless the Thread Execution
125233 Scheduling option is supported.

125234 The standard developers considered the issue of scheduling with respect to the queuing of
125235 threads blocked on a read-write lock. The question turned out to be whether POSIX.1-2017
125236 should require priority scheduling of read-write locks for threads whose execution scheduling
125237 policy is priority-based (for example, SCHED_FIFO or SCHED_RR). There are tradeoffs
125238 between priority scheduling, the amount of concurrency achievable among readers, and the
125239 prevention of writer and/or reader starvation.

125240 For example, suppose one or more readers hold a read-write lock and the following threads
125241 request the lock in the listed order:

```
125242 pthread_rwlock_wrlock() - Low priority thread writer_a
125243 pthread_rwlock_rdlock() - High priority thread reader_a
125244 pthread_rwlock_rdlock() - High priority thread reader_b
125245 pthread_rwlock_rdlock() - High priority thread reader_c
```

125246 When the lock becomes available, should *writer_a* block the high priority readers? Or, suppose a
125247 read-write lock becomes available and the following are queued:

```
125248 pthread_rwlock_rdlock() - Low priority thread reader_a
125249 pthread_rwlock_rdlock() - Low priority thread reader_b
125250 pthread_rwlock_rdlock() - Low priority thread reader_c
125251 pthread_rwlock_wrlock() - Medium priority thread writer_a
125252 pthread_rwlock_rdlock() - High priority thread reader_d
```

125253 If priority scheduling is applied then *reader_d* would acquire the lock and *writer_a* would block
125254 the remaining readers. But should the remaining readers also acquire the lock to increase
125255 concurrency? The solution adopted takes into account that when the Thread Execution
125256 Scheduling option is supported, high priority threads may in fact starve low priority threads
125257 (the application developer is responsible in this case for designing the system in such a way that
125258 this starvation is avoided). Therefore, POSIX.1-2017 specifies that high priority readers take
125259 precedence over lower priority writers. However, to prevent writer starvation from threads of
125260 the same or lower priority, writers take precedence over readers of the same or lower priority.

125261 Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high
125262 priority writer is forced to wait for multiple readers, for example, it is not clear which subset of
125263 the readers should inherit the writer's priority. Furthermore, the internal data structures that
125264 record the inheritance must be accessible to all readers, and this implies some sort of
125265 serialization that could negate any gain in parallelism achieved through the use of multiple
125266 readers in the first place. Finally, existing practice does not support the use of priority
125267 inheritance for read-write locks. Therefore, no specification of priority inheritance or priority
125268 ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can
125269 always be obtained through the use of mutexes.

125270 **Comparison to `fcntl()` Locks**

125271 The read-write locks and the `fcntl()` locks in POSIX.1-2017 share a common goal: increasing
 125272 concurrency among readers, thus increasing throughput and decreasing delay.

125273 However, the read-write locks have two features not present in the `fcntl()` locks. First, under
 125274 priority scheduling, read-write locks are granted in priority order. Second, also under priority
 125275 scheduling, writer starvation is prevented by giving writers preference over readers of equal or
 125276 lower priority.

125277 Also, read-write locks can be used in systems lacking a file system, such as those conforming to
 125278 the minimal realtime system profile of IEEE Std 1003.13-1998.

125279 **History of Resolution Issues**

125280 Based upon some balloting objections, early drafts specified the behavior of threads waiting on a
 125281 read-write lock during the execution of a signal handler, as if the thread had not called the lock
 125282 operation. However, this specified behavior would require implementations to establish
 125283 internal signal handlers even though this situation would be rare, or never happen for many
 125284 programs. This would introduce an unacceptable performance hit in comparison to the little
 125285 additional functionality gained. Therefore, the behavior of read-write locks and signals was
 125286 reverted back to its previous mutex-like specification.

125287 *B.2.9.7 Thread Interactions with Regular File Operations*

125288 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0028 [498] is applied.

125289 *B.2.9.8 Use of Application-Managed Thread Stacks*

125290 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/8 is applied, adding this new section. It
 125291 was added to make it clear that the current standard does not allow an application to determine
 125292 when a stack can be reclaimed. This may be addressed in a future version.

125293 *B.2.9.9 Synchronization Object Copies and Alternative Mappings*

125294 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0029 [972] is applied.

125295 **B.2.10 Sockets**

125296 The base document for the sockets interfaces in POSIX.1-2017 is the XNS, Issue 5.2 specification.
 125297 This was primarily chosen as it aligns with IPv6. Additional material has been added from
 125298 IEEE Std 1003.1g-2000, notably socket concepts, raw sockets, the `pselect()` function, the
 125299 `socketmark()` function, and the `<sys/select.h>` header.

125300 *B.2.10.1 Address Families*

125301 There is no additional rationale provided for this section.

125302 B.2.10.2 Addressing

125303 There is no additional rationale provided for this section.

125304 B.2.10.3 Protocols

125305 There is no additional rationale provided for this section.

125306 B.2.10.4 Routing

125307 There is no additional rationale provided for this section.

125308 B.2.10.5 Interfaces

125309 There is no additional rationale provided for this section.

125310 B.2.10.6 Socket Types

125311 The type **socklen_t** was invented to cover the range of implementations seen in the field. The
125312 intent of **socklen_t** is to be the type for all lengths that are naturally bounded in size; that is, that
125313 they are the length of a buffer which cannot sensibly become of massive size: network addresses,
125314 host names, string representations of these, ancillary data, control messages, and socket options
125315 are examples. Truly boundless sizes are represented by **size_t** as in *read()*, *write()*, and so on.

125316 All **socklen_t** types were originally (in BSD UNIX) of type **int**. During the development of
125317 POSIX.1-2017, it was decided to change all buffer lengths to **size_t**, which appears at face value
125318 to make sense. When dual mode 32/64-bit systems came along, this choice unnecessarily
125319 complicated system interfaces because **size_t** (with **long**) was a different size under ILP32 and
125320 LP64 models. Reverting to **int** would have happened except that some implementations had
125321 already shipped 64-bit-only interfaces. The compromise was a type which could be defined to be
125322 any size by the implementation: **socklen_t**.

125323 B.2.10.7 Socket I/O Mode

125324 There is no additional rationale provided for this section.

125325 B.2.10.8 Socket Owner

125326 There is no additional rationale provided for this section.

125327 B.2.10.9 Socket Queue Limits

125328 There is no additional rationale provided for this section.

125329 B.2.10.10 Pending Error

125330 There is no additional rationale provided for this section.

125331 *B.2.10.11 Socket Receive Queue*

125332 There is no additional rationale provided for this section.

125333 *B.2.10.12 Socket Out-of-Band Data State*

125334 There is no additional rationale provided for this section.

125335 *B.2.10.13 Connection Indication Queue*

125336 There is no additional rationale provided for this section.

125337 *B.2.10.14 Signals*

125338 There is no additional rationale provided for this section.

125339 *B.2.10.15 Asynchronous Errors*

125340 There is no additional rationale provided for this section.

125341 *B.2.10.16 Use of Options*

125342 There is no additional rationale provided for this section.

125343 *B.2.10.17 Use of Sockets for Local UNIX Connections*

125344 There is no additional rationale provided for this section.

125345 *B.2.10.18 Use of Sockets over Internet Protocols*

125346 A raw socket allows privileged users direct access to a protocol; for example, raw access to the
125347 IP and ICMP protocols is possible through raw sockets. Raw sockets are intended for
125348 knowledgeable applications that wish to take advantage of some protocol feature not directly
125349 accessible through the other sockets interfaces.

125350 *B.2.10.19 Use of Sockets over Internet Protocols Based on IPv4*

125351 There is no additional rationale provided for this section.

125352 *B.2.10.20 Use of Sockets over Internet Protocols Based on IPv6*

125353 The Open Group Base Resolution bwg2001-012 is applied, clarifying that IPv6 implementations
125354 are required to support use of AF_INET6 sockets over IPv4.

125355 **B.2.11 Tracing**

125356 The organization of the tracing rationale differs from the traditional rationale in that this tracing
 125357 rationale text is written against the trace interface as a whole, rather than against the individual
 125358 components of the trace interface or the normative section in which those components are
 125359 defined. Therefore the sections below do not parallel the sections of normative text in
 125360 POSIX.1-2017.

125361 *B.2.11.1 Objectives*

125362 The intended uses of tracing are application-system debugging during system development, as a
 125363 “flight recorder” for maintenance of fielded systems, and as a performance measurement tool.
 125364 In all of these intended uses, the vendor-supplied computer system and its software are, for this
 125365 discussion, assumed error-free; the intent being to debug the user-written and/or third-party
 125366 application code, and their interactions. Clearly, problems with the vendor-supplied system and
 125367 its software will be uncovered from time to time, but this is a byproduct of the primary activity,
 125368 debugging user code.

125369 Another need for defining a trace interface in POSIX stems from the objective to provide an
 125370 efficient portable way to perform benchmarks. Existing practice shows that such interfaces are
 125371 commonly used in a variety of systems but with little commonality. As part of the benchmarking
 125372 needs, two aspects within the trace interface must be considered.

125373 The first, and perhaps more important one, is the qualitative aspect.

125374 The second is the quantitative aspect.

125375 **Qualitative Aspect**

125376 To better understand this aspect, let us consider an example. Suppose that you want to
 125377 organize a number of actions to be performed during the day. Some of these actions are
 125378 known at the beginning of the day. Some others, which may be more or less important,
 125379 will be triggered by reading your mail. During the day you will make some phone calls
 125380 and synchronously receive some more information. Finally you will receive asynchronous
 125381 phone calls that also will trigger actions. If you, or somebody else, examines your day at
 125382 work, you, or he, can discover that you have not efficiently organized your work. For
 125383 instance, relative to the phone calls you made, would it be preferable to make some of
 125384 these early in the morning? Or to delay some others until the end of the day? Relative to
 125385 the phone calls you have received, you might find that somebody you called in the
 125386 morning has called you 10 times while you were performing some important work. To
 125387 examine, afterwards, your day at work, you record in sequence all the trace events relative
 125388 to your work. This should give you a chance of organizing your next day at work.

125389 This is the qualitative aspect of the trace interface. The user of a system needs to keep a
 125390 trace of particular points the application passes through, so that he can eventually make
 125391 some changes in the application and/or system configuration, to give the application a
 125392 chance of running more efficiently.

125393 **Quantitative Aspect**

125394 This aspect concerns primarily realtime applications, where missed deadlines can be
 125395 undesirable. Although there are, in POSIX.1-2017, some interfaces useful for such
 125396 applications (timeouts, execution time monitoring, and so on), there are no APIs to aid in
 125397 the tuning of a realtime application’s behavior (**timespec** in timeouts, length of message
 125398 queues, duration of driver interrupt service routine, and so on). The tuning of an
 125399 application needs a means of recording timestamped important trace events during
 125400 execution in order to analyze offline, and eventually, to tune some realtime features

125401 (redesign the system with less functionalities, readjust timeouts, redesign driver interrupts,
125402 and so on).

125403 Detailed Objectives

125404 Objectives were defined to build the trace interface and are kept for historical interest. Although
125405 some objectives are not fully respected in this trace interface, the concept of the POSIX trace
125406 interface assumes the following points:

- 125407 1. It must be possible to trace both system and user trace events concurrently.
- 125408 2. It must be possible to trace per-process trace events and also to trace system trace events
125409 which are unrelated to any particular process. A per-process trace event is either user-
125410 initiated or system-initiated.
- 125411 3. It must be possible to control tracing on a per-process basis from either inside or outside
125412 the process.
- 125413 4. It must be possible to control tracing on a per-thread basis from inside the enclosing
125414 process.
- 125415 5. Trace points must be controllable by trace event type ID from inside and outside of the
125416 process. Multiple trace points can have the same trace event type ID, and will be
125417 controlled jointly.
- 125418 6. Recording of trace events is dependent on both trace event type ID and the
125419 process/thread. Both must be enabled in order to record trace events. System trace events
125420 may or may not be handled differently.
- 125421 7. The API must not mandate the ability to control tracing for more than one process at the
125422 same time.
- 125423 8. There is no objective for trace control on anything bigger than a process; for example,
125424 group or session.
- 125425 9. Trace propagation and control:
 - 125426 a. Trace propagation across *fork()* is optional; the default is to not trace a child
125427 process.
 - 125428 b. Trace control must span *pthread_create()* operations; that is, if a process is being
125429 traced, any thread will be traced as well if this thread allows tracing. The default is
125430 to allow tracing.
- 125431 10. Trace control must not span *exec* or *posix_spawn()* operations.
- 125432 11. A triggering API is not required. The triggering API is the ability to command or stop
125433 tracing based on the occurrence of a specific trace event other than a
125434 POSIX_TRACE_START trace event or a POSIX_TRACE_STOP trace event.
- 125435 12. Trace log entries must have timestamps of implementation-defined resolution.
125436 Implementations are exhorted to support at least microsecond resolution. When a trace
125437 log entry is retrieved, it must have timestamp, PC address, PID, and TID of the entity that
125438 generated the trace event.
- 125439 13. Independently developed code should be able to use trace facilities without coordination
125440 and without conflict.
- 125441 14. Even if the trace points in the trace calls are not unique, the trace log entries (after any
125442 processing) must be uniquely identified as to trace point.

- 125443 15. There must be a standard API to read the trace stream.
- 125444 16. The format of the trace stream and the trace log is opaque and unspecified.
- 125445 17. It must be possible to read a completed trace, if recorded on some suitable non-volatile
125446 storage, even subsequent to a power cycle or subsequent cold boot of the system.
- 125447 18. Support of analysis of a trace log while it is being formed is implementation-defined.
- 125448 19. The API must allow the application to write trace stream identification information into
125449 the trace stream and to be able to retrieve it, without it being overwritten by trace entries,
125450 even if the trace stream is full.
- 125451 20. It must be possible to specify the destination of trace data produced by trace events.
- 125452 21. It must be possible to have different trace streams, and for the tracing enabled by one
125453 trace stream to be completely independent of the tracing of another trace stream.
- 125454 22. It must be possible to trace events from threads in different CPUs.
- 125455 23. The API must support one or more trace streams per-system, and one or more trace
125456 streams per-process, up to an implementation-defined set of per-system and per-process
125457 maximums.
- 125458 24. It must be possible to determine the order in which the trace events happened, without
125459 necessarily depending on the clock, up to an implementation-defined time resolution.
- 125460 25. For performance reasons, the trace event point call(s) must be implementable as a macro
125461 (see the ISO POSIX-1: 1996 standard, 1.3.4, Statement 2).
- 125462 26. POSIX.1-2017 must not define the trace points which a conforming system must
125463 implement, except for trace points used in the control of tracing.
- 125464 27. The APIs must be thread-safe, and trace points should be lock-free (that is, not require a
125465 lock to gain exclusive access to some resource).
- 125466 28. The user-provided information associated with a trace event is variable-sized, up to some
125467 maximum size.
- 125468 29. Bounds on record and trace stream sizes:
- 125469 a. The API must permit the application to declare the upper bounds on the length of
125470 an application data record. The system must return the limit it used. The limit used
125471 may be smaller than requested.
- 125472 b. The API must permit the application to declare the upper bounds on the size of
125473 trace streams. The system must return the limit it used. The limit used may be
125474 different, either larger or smaller, than requested.
- 125475 30. The API must be able to pass any fundamental data type, and a structured data type
125476 composed only of fundamental types. The API must be able to pass data by reference,
125477 given only as an address and a length. Fundamental types are the POSIX.1 types (see the
125478 **<sys/types.h>** header) plus those defined in the ISO C standard.
- 125479 31. The API must apply the POSIX notions of ownership and permission to recorded trace
125480 data, corresponding to the sources of that data.

125481 **Comments on Objectives**

125482 **Note:** In the following comments, numbers in square brackets refer to the above objectives.

125483 It is necessary to be able to obtain a trace stream for a complete activity. Thus there is a
125484 requirement to be able to trace both application and system trace events. A per-process trace
125485 event is either user-initiated, like the *write()* function, or system-initiated, like a timer expiration.
125486 There is also a need to be able to trace the activity of an entire process even when it has threads
125487 in multiple CPUs. To avoid excess trace activity, it is necessary to be able to control tracing on a
125488 trace event type basis.

125489 [Objectives 1,2,5,22]

125490 There is a need to be able to control tracing on a per-process basis, both from inside and outside
125491 the process; that is, a process can start a trace activity on itself or any other process. There is also
125492 the perceived need to allow the definition of a maximum number of trace streams per system.

125493 [Objectives 3,23]

125494 From within a process, it is necessary to be able to control tracing on a per-thread basis. This
125495 provides an additional filtering capability to keep the amount of traced data to a minimum. It
125496 also allows for less ambiguity as to the origin of trace events. It is recognized that thread-level
125497 control is only valid from within the process itself. It is also desirable to know the maximum
125498 number of trace streams per process that can be started. The API should not require thread
125499 synchronization or mandate priority inversions that would cause the thread to block. However,
125500 the API must be thread-safe.

125501 [Objectives 4,23,24,27]

125502 There was no perceived objective to control tracing on anything larger than a process; for
125503 example, a group or session. Also, the ability to start or stop a trace activity on multiple
125504 processes atomically may be very difficult or cumbersome in some implementations.

125505 [Objectives 6,8]

125506 It is also necessary to be able to control tracing by trace event type identifier, sometimes called a
125507 trace hook ID. However, there is no mandated set of system trace events, since such trace points
125508 are implementation-defined. The API must not require from the operating system facilities that
125509 are not standard.

125510 [Objectives 6,26]

125511 Trace control must span *fork()* and *pthread_create()*. If not, there will be no way to ensure that an
125512 application's activity is entirely traced. The newly forked child would not be able to turn on its
125513 tracing until after it obtained control after the fork, and trace control externally would be even
125514 more problematic.

125515 [Objective 9]

125516 Since *exec* and *posix_spawn()* represent a complete change in the execution of a task (a new
125517 program), trace control need not persist over an *exec* or *posix_spawn()*.

125518 [Objective 10]

125519 Where trace activities are started on multiple processes, these trace activities should not interfere
125520 with each other.

125521 [Objective 21]

125522 There is no need for a triggering objective, primarily for performance reasons; see also [Section](#)
125523 [B.2.11.8](#) (on page 3687), rationale on triggering.

125524 [Objective 11]

125525 It must be possible to determine the origin of each traced event. The process and thread
125526 identifiers for each trace event are needed. Also there was a perceived need for a user-specifiable
125527 origin, but it was felt that this would create too much overhead.

125528 [Objectives 12,14]

- 125529 An allowance must be made for trace points to come embedded in software components from
125530 several different sources and vendors without requiring coordination.
125531 [Objective 13]
- 125532 There is a requirement to be able to uniquely identify trace points that may have the same trace
125533 stream identifier. This is only necessary when a trace report is produced.
125534 [Objectives 12,14]
- 125535 Tracing is a very performance-sensitive activity, and will therefore likely be implemented at a
125536 low level within the system. Hence the interface must not mandate any particular buffering or
125537 storage method. Therefore, a standard API is needed to read a trace stream. Also the interface
125538 must not mandate the format of the trace data, and the interface must not assume a trace storage
125539 method. Due to the possibility of a monolithic kernel and the possible presence of multiple
125540 processes capable of running trace activities, the two kinds of trace events may be stored in two
125541 separate streams for performance reasons. A mandatory dump mechanism, common in some
125542 existing practice, has been avoided to allow the implementation of this set of functions on small
125543 realtime profiles for which the concept of a file system is not defined. The trace API calls should
125544 be implemented as macros.
125545 [Objectives 15,16,25,30]
- 125546 Since a trace facility is a valuable service tool, the output (or log) of a completed trace stream
125547 that is written to permanent storage must be readable on other systems of the type that
125548 produced the trace log. Note that there is no objective to be able to interpret a trace log that was
125549 not successfully completed.
125550 [Objectives 17,18,19]
- 125551 For trace streams written to permanent storage, a way to specify the destination of the trace
125552 stream is needed.
125553 [Objective 20]
- 125554 There is a requirement to be able to depend on the ordering of trace events up to some
125555 implementation-defined time interval. For example, there is a need to know the time period
125556 during which, if trace events are closer together, their ordering is unspecified. Events that occur
125557 within an interval smaller than this resolution may or may not be read back in the correct order.
125558 [Objective 24]
- 125559 The application should be able to know how much data can be traced. When trace event types
125560 can be filtered, the application should be able to specify the approximate maximum amount of
125561 data that will be traced in a trace event so resources can be more efficiently allocated.
125562 [Objectives 28,29]
- 125563 Users should not be able to trace data to which they would not normally have access. System
125564 trace events corresponding to a process/thread should be associated with the ownership of that
125565 process/thread.
125566 [Objective 31]

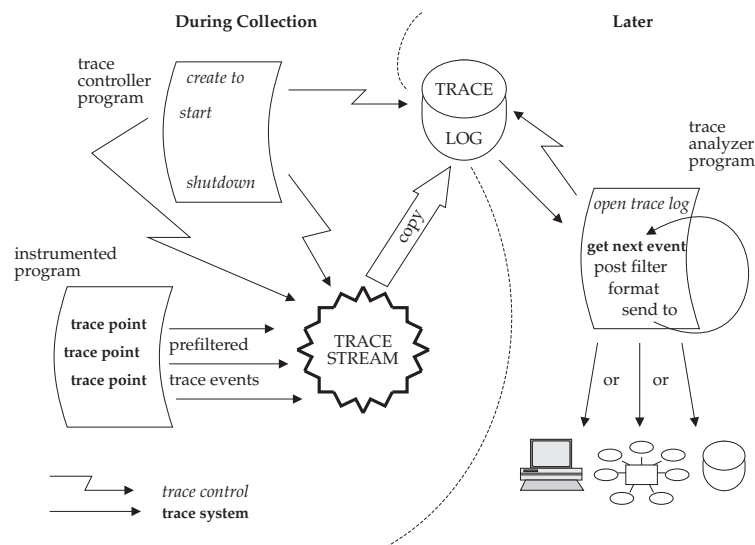
125567 B.2.11.2 Trace Model

125568 **Introduction**

125569 The model is based on two base entities: the "Trace Stream" and the "Trace Log", and a recorded
 125570 unit called the "Trace Event". The possibility of using Trace Streams and Trace Logs separately
 125571 gives two use dimensions and solves both the performance issue and the full-information
 125572 system issue. In the case of a trace stream without log, specific information, although reduced in
 125573 quantity, is required to be registered, in a possibly small realtime system, with as little overhead
 125574 as possible. The Trace Log option has been added for small realtime systems. In the case of a
 125575 trace stream with log, considerable complex application-specific information needs to be
 125576 collected.

125577 **Trace Model Description**

125578 The trace model can be examined for three different subfunctions: Application Instrumentation,
 125579 Trace Operation Control, and Trace Analysis.

125580 **Figure B-2** Trace System Overview: for Offline Analysis

125581 Each of these subfunctions requires specific characteristics of the trace mechanism API.

125582 **Application Instrumentation**

125583 When instrumenting an application, the programmer is not concerned about the future use
 125584 of the trace events in the trace stream or the trace log, the full policy of the trace stream, or
 125585 the eventual pre-filtering of trace events. But he is concerned about the correct
 125586 determination of the specific trace event type identifier, regardless of how many
 125587 independent libraries are used in the same user application; see [Figure B-2](#) and [Figure B-3](#)
 125588 (on page 3670).

125589 This trace API provides the necessary operations to accomplish this subfunction. This is
 125590 done by providing functions to associate a programmer-defined name with an
 125591 implementation-defined trace event type identifier (see the `posix_trace_eventid_open()`
 125592 function), and to send this trace event into a potential trace stream (see the
 125593 `posix_trace_event()` function).

125594 Trace Operation Control

125595 When controlling the recording of trace events in a trace stream, the programmer is
 125596 concerned with the correct initialization of the trace mechanism (that is, the sizing of the
 125597 trace stream), the correct retention of trace events in a permanent storage, the correct
 125598 dynamic recording of trace events, and so on.

125599 This trace API provides the necessary material to permit this efficiently. This is done by
 125600 providing functions to initialize a new trace stream, and optionally a trace log:

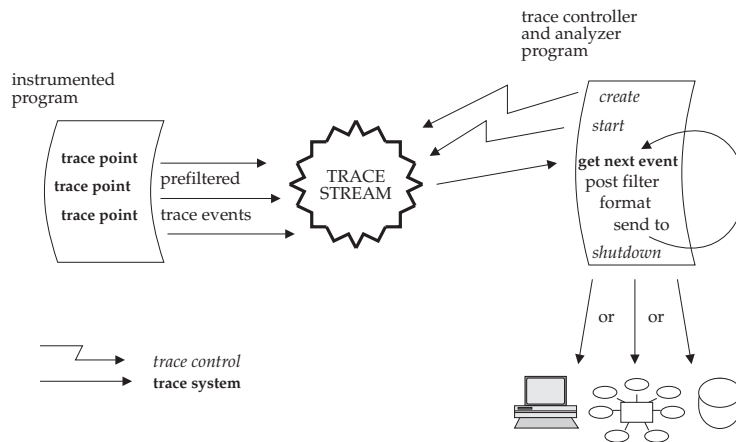
- 125601 † **Trace Stream Attributes Object Initialization** (see *posix_trace_attr_init()*)
- 125602 † **Functions to Retrieve or Set Information About a Trace Stream** (see
 125603 *posix_trace_attr_getgenversion()*)
- 125604 † **Functions to Retrieve or Set the Behavior of a Trace Stream** (see
 125605 *posix_trace_attr_getinherited()*)
- 125606 † **Functions to Retrieve or Set Trace Stream Size Attributes** (see
 125607 *posix_trace_attr_getmaxusereventsize()*)
- 125608 † **Trace Stream Initialization, Flush, and Shutdown from a Process** (see
 125609 *posix_trace_create()*)
- 125610 † **Clear Trace Stream and Trace Log** (see *posix_trace_clear()*)

125611 To select the trace event types that are to be traced:

- 125612 † **Allocate Trace Event Type Identifier** (see *posix_trace_trid_eventid_open()*)
- 125613 † **Iterate over a Mapping of Trace Event Type** (see *posix_trace_eventtypelist_getnext_id()*)
- 125614 † **Allocate Trace Event Type Sets** (see *posix_trace_eventset_empty()*)
- 125615 † **Set Filter of an Initialized Trace Stream** (see *posix_trace_set_filter()*)

125616 To control the execution of an active trace stream:

- 125617 † **Trace Start and Stop** (see *posix_trace_start()*)
- 125618 † **Functions to Retrieve the Trace Attributes or Trace Statuses** (see
 125619 *posix_trace_get_attr()*)



125620 **Figure B-3** Trace System Overview: for Online Analysis

125621 Trace Analysis

125622 Once correctly recorded, on permanent storage or not, an ultimate activity consists of the
125623 analysis of the recorded information. If the recorded data is on permanent storage, a
125624 specific open operation is required to associate a trace stream to a trace log.

125625 The first intent of the group was to request the presence of a system identification structure
125626 in the trace stream attribute. This was, for the application, to allow some portable way to
125627 process the recorded information. However, there is no requirement that the **utsname**
125628 structure, on which this system identification was based, be portable from one machine to
125629 another, so the contents of the attribute cannot be interpreted correctly by an application
125630 conforming to POSIX.1-2017.

125631 This modification has been incorporated and requests that some unspecified information
125632 be recorded in the trace log in order to fail opening it if the analysis process and the
125633 controller process were running in different types of machine, but does not request that
125634 this information be accessible to the application. This modification has implied a
125635 modification in the *posix_trace_open()* function error code returns.

125636 This trace API provides functions to:

125637 † ~~to~~ fact trace stream identification attributes (see *posix_trace_attr_getgenversion()*)

125638 † ~~to~~ fact trace stream behavior attributes (see *posix_trace_attr_getinherited()*)

125639 † ~~to~~ fact trace event, stream, and log size attributes (see
125640 *posix_trace_attr_getmaxusereventsized()*)

125641 † ~~to~~ look up trace event type names (see *posix_trace_eventid_get_name()*)

125642 † ~~to~~ iterate over trace event type identifiers (see *posix_trace_eventtypelist_getnext_id()*)

125643 † ~~to~~ open, rewind, and close a trace log (see *posix_trace_open()*)

125644 † ~~to~~ read trace stream attributes and status (see *posix_trace_get_attr()*)

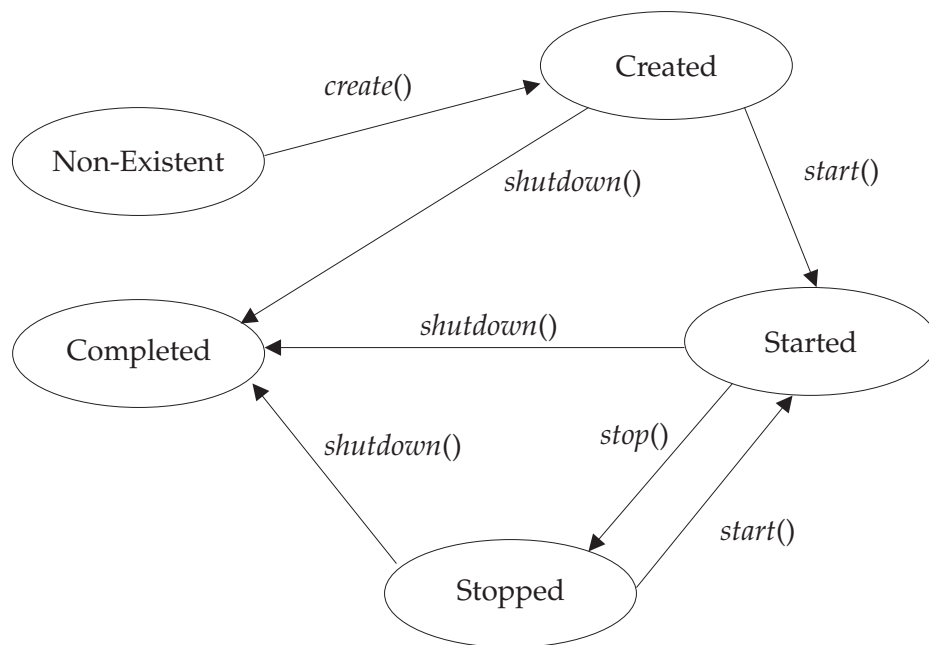
125645 † ~~to~~ read trace events (see *posix_trace_getnext_event()*)

125646 Due to the following two reasons:

- 125647 1. The requirement that the trace system must not add unacceptable overhead to the traced
125648 process and so that the trace event point execution must be fast
- 125649 2. The traced application does not care about tracing errors

125650 the trace system cannot return any internal error to the application. Internal error conditions can
125651 range from unrecoverable errors that will force the active trace stream to abort, to small errors
125652 that can affect the quality of tracing without aborting the trace stream. The group decided to
125653 define a system trace event to report to the analysis process such internal errors. It is not the
125654 intention of POSIX.1-2017 to require an implementation to report an internal error that corrupts
125655 or terminates tracing operation. The implementor is free to decide which internal documented
125656 errors, if any, the trace system is able to report.

125657

States of a Trace Stream

125658

Figure B-4 Trace System Overview: States of a Trace Stream

125659
 125660
 125661
 125662
 125663
 125664
 125665
 125666
 125667

Figure B-4 shows the different states an active trace stream passes through. After the *posix_trace_create()* function call, a trace stream becomes CREATED and a trace stream is associated for the future collection of trace events. The status of the trace stream is POSIX_TRACE_SUSPENDED. The state becomes STARTED after a call to the *posix_trace_start()* function, and the status becomes POSIX_TRACE_RUNNING. In this state, all trace events that are not filtered out will be stored into the trace stream. After a call to *posix_trace_stop()*, the trace stream becomes STOPPED (and the status POSIX_TRACE_SUSPENDED). In this state, no new trace events will be recorded in the trace stream, but previously recorded trace events may continue to be read.

125668
 125669
 125670
 125671
 125672

After a call to *posix_trace_shutdown()*, the trace stream is in the state COMPLETED. The trace stream no longer exists but, if the Trace Log option is supported, all the information contained in it has been logged. If a log object has not been associated with the trace stream at the creation, it is the responsibility of the trace controller process to not shut the trace stream down while trace events remain to be read in the stream.

125673

Tracing All Processes

125674
 125675
 125676
 125677

Some implementations have a tracing subsystem with the ability to trace all processes. This is useful to debug some types of device drivers such as those for ATM or X25 adapters. These types of adapters are used by several independent processes, that are not issued from the same process.

125678
 125679
 125680
 125681

The POSIX trace interface does not define any constant or option to create a trace stream tracing all processes. POSIX.1 does not prevent this type of implementation and an implementor is free to add this capability. Nevertheless, the trace interface allows tracing of all the system trace events and all the processes issued from the same process.

125682 If such a tracing system capability has to be implemented, when a trace stream is created, it is
 125683 recommended that a constant named `POSIX_TRACE_ALLPROC` be used instead of the process
 125684 identifier in the argument of the `posix_trace_create()` or `posix_trace_create_withlog()` function. A
 125685 possible value for `POSIX_TRACE_ALLPROC` may be `-1` instead of a real process identifier.

125686 The implementor has to be aware that there is some impact on the tracing behavior as defined in
 125687 the POSIX trace interface. For example:

125688 If the default value for the inheritance attribute is set to
 125689 `POSIX_TRACE_CLOSE_FOR_CHILD`, the implementation has to stop tracing for the child
 125690 process.

125691 The trace controller which is creating this type of trace stream must have the appropriate
 125692 privilege to trace all the processes.

125693 **Trace Storage**

125694 The model is based on two types of trace events: system trace events and user-defined trace
 125695 events. The internal representation of trace events is implementation-defined, and so the
 125696 implementor is free to choose the more suitable, practical, and efficient way to design the
 125697 internal management of trace events. For the timestamping operation, the model does not
 125698 impose the `CLOCK_REALTIME` or any other clock. The buffering allocation and operation
 125699 follow the same principle. The implementor is free to use one or more buffers to record trace
 125700 events; the interface assumes only a logical trace stream of sequentially recorded trace events.
 125701 Regarding flushing of trace events, the interface allows the definition of a trace log object which
 125702 typically can be a file. But the group was also aware of defining functions to permit the use of
 125703 this interface in small realtime systems, which may not have general file system capabilities. For
 125704 instance, the three functions `posix_trace_getnext_event()` (blocking),
 125705 `posix_trace_timedgetnext_event()` (blocking with timeout), and `posix_trace_trygetnext_event()` (non-
 125706 blocking) are proposed to read the recorded trace events.

125707 The policy to be used when the trace stream becomes full also relies on common practice:

125708 For an active trace stream, the `POSIX_TRACE_LOOP` trace stream policy permits
 125709 automatic overrun (overwrite of oldest trace events) while waiting for some user-defined
 125710 condition to cause tracing to stop. By contrast, the `POSIX_TRACE_UNTIL_FULL` trace
 125711 stream policy requires the system to stop tracing when the trace stream is full. However, if
 125712 the trace stream that is full is at least partially emptied by a call to the `posix_trace_flush()`
 125713 function or by calls to the `posix_trace_getnext_event()` function, the trace system will
 125714 automatically resume tracing.

125715 If the Trace Log option is supported, the operation of the `POSIX_TRACE_FLUSH` policy is
 125716 an extension of the `POSIX_TRACE_UNTIL_FULL` policy. The automatic free operation (by
 125717 flushing to the associated trace log) is added.

125718 If a log is associated with the trace stream and this log is a regular file, these policies also
 125719 apply for the log. One more policy, `POSIX_TRACE_APPEND`, is defined to allow
 125720 indefinite extension of the log. Since the log destination can be any device or pseudo-
 125721 device, the implementation may not be able to manipulate the destination as required by
 125722 POSIX.1-2017. For this reason, the behavior of the log full policy may be unspecified
 125723 depending on the trace log type.

125724 The current trace interface does not define a service to preallocate space for a trace log file,
 125725 because this space can be preallocated by means of a call to the `posix_fallocate()` function.
 125726 This function could be called after the file has been opened, but before the trace stream is
 125727 created. The `posix_fallocate()` function ensures that any required storage for regular file data
 125728 is allocated on the file system storage media. If `posix_fallocate()` returns successfully,

125729 subsequent writes to the specified file data will not fail due to the lack of free space on the
 125730 file system storage media. Besides trace events, a trace stream also includes trace attributes
 125731 and the mapping from trace event names to trace event type identifiers. The implementor
 125732 is free to choose how to store the trace attributes and the trace event type map, but must
 125733 ensure that this information is not lost when a trace stream overrun occurs.

125734 B.2.11.3 Trace Programming Examples

125735 Several programming examples are presented to show the code of the different possible
 125736 subfunctions using a trace subsystem. All these programs need to include the `<trace.h>` header.
 125737 In the examples shown, error checking is omitted for more simplicity.

125738 Trace Operation Control

125739 These examples show the creation of a trace stream for another process; one which is already
 125740 trace instrumented. All the default trace stream attributes are used to simplify programming in
 125741 the first example. The second example shows more possibilities.

125742 First Example

```

125743 /* Caution. Error checks omitted */
125744 {
125745     trace_attr_t attr;
125746     pid_t pid = traced_process_pid;
125747     int fd;
125748     trace_id_t trid;
125749
125750     - - - - -
125751     /* Initialize trace stream attributes */
125752     posix_trace_attr_init(&attr);
125753     /* Open a trace log */
125754     fd=open("/tmp/mytracelog",...);
125755     /*
125756      * Create a new trace associated with a log
125757      * and with default attributes
125758      */
125759     posix_trace_create_withlog(pid, &attr, fd, &trid);
125760
125761     /* Trace attribute structure can now be destroyed */
125762     posix_trace_attr_destroy(&attr);
125763     /* Start of trace event recording */
125764     posix_trace_start(trid);
125765     - - - - -
125766     /* Duration of tracing */
125767     - - - - -
125768     /* Stop and shutdown of trace activity */
125769     posix_trace_shutdown(trid);
125770     - - - - -
125771 }
  
```

125772 **Second Example**

125773 Between the initialization of the trace stream attributes and the creation of the trace stream, these
 125774 trace stream attributes may be modified; see [Trace Stream Attribute Manipulation](#) (on page
 125775 3678) for a specific programming example. Between the creation and the start of the trace
 125776 stream, the event filter may be set; after the trace stream is started, the event filter may be
 125777 changed. The setting of an event set and the change of a filter is shown in [Create a Trace Event
 125778 Type Set and Change the Trace Event Type Filter](#) (on page 3679).

```

125779 /* Caution. Error checks omitted */
125780 {
125781     trace_attr_t attr;
125782     pid_t pid = traced_process_pid;
125783     int fd;
125784     trace_id_t trid;
125785     - - - - -
125786     /* Initialize trace stream attributes */
125787     posix_trace_attr_init(&attr);
125788     /* Attr default may be changed at this place; see example */
125789     - - - - -
125790     /* Create and open a trace log with R/W user access */
125791     fd=open("/tmp/mytracelog",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
125792     /* Create a new trace associated with a log */
125793     posix_trace_create_withlog(pid, &attr, fd, &trid);
125794     /*
125795      * If the Trace Filter option is supported
125796      * trace event type filter default may be changed at this place;
125797      * see example about changing the trace event type filter
125798      */
125799     posix_trace_start(trid);
125800     - - - - -
125801     /*
125802      * If you have an uninteresting part of the application
125803      * you can stop temporarily.
125804      *
125805      * posix_trace_stop(trid);
125806      * - - - - -
125807      * - - - - -
125808      * posix_trace_start(trid);
125809      */
125810     - - - - -
125811     /*
125812      * If the Trace Filter option is supported
125813      * the current trace event type filter can be changed
125814      * at any time (see example about how to set
125815      * a trace event type filter)
125816      */
125817     - - - - -
125818     /* Stop the recording of trace events */
125819     posix_trace_stop(trid);
125820     /* Shutdown the trace stream */
125821     posix_trace_shutdown(trid);
125822     /*
  
```

```

125823     * Destroy trace stream attributes; attr structure may have
125824     * been used during tracing to fetch the attributes
125825     */
125826     posix_trace_attr_destroy(&attr);
125827     - - - - -
125828 }

```

125829 **Application Instrumentation**

125830 This example shows an instrumented application. The code is included in a block of instructions,
 125831 perhaps a function from a library. Possibly in an initialization part of the instrumented
 125832 application, two user trace event names are mapped to two trace event type identifiers
 125833 (function `posix_trace_eventid_open()`). Then two trace points are programmed.

```

125834 /* Caution. Error checks omitted */
125835 {
125836     trace_event_id_t eventid1, eventid2;
125837     - - - - -
125838     /* Initialization of two trace event type ids */
125839     posix_trace_eventid_open("my_first_event",&eventid1);
125840     posix_trace_eventid_open("my_second_event",&eventid2);
125841     - - - - -
125842     - - - - -
125843     - - - - -
125844     /* Trace point */
125845     posix_trace_event(eventid1,NULL,0);
125846     - - - - -
125847     /* Trace point */
125848     posix_trace_event(eventid2,NULL,0);
125849     - - - - -
125850 }

```

125851 **Trace Analyzer**

125852 This example shows the manipulation of a trace log resulting from the dumping of a completed
 125853 trace stream. All the default attributes are used to simplify programming, and data associated
 125854 with a trace event is not shown in the first example. The second example shows more
 125855 possibilities.

125856 **First Example**

```

125857 /* Caution. Error checks omitted */
125858 {
125859     int fd;
125860     trace_id_t trid;
125861     posix_trace_event_info trace_event;
125862     char trace_event_name[TRACE_EVENT_NAME_MAX];
125863     int return_value;
125864     size_t returndatasize;
125865     int lost_event_number;
125866     - - - - -
125867     /* Open an existing trace log */
125868     fd=open("/tmp/tracelog", O_RDONLY);

```

```

125869     /* Open a trace stream on the open log */
125870     posix_trace_open(fd, &trid);
125871     /* Read a trace event */
125872     posix_trace_getnext_event(trid, &trace_event,
125873         NULL, 0, &returndatasize,&return_value);

125874     /* Read and print all trace event names out in a loop */
125875     while (return_value == NULL)
125876     {
125877         /*
125878          * Get the name of the trace event associated
125879          * with trid trace ID
125880          */
125881         posix_trace_eventid_get_name(trid, trace_event.event_id,
125882             trace_event_name);
125883         /* Print the trace event name out */
125884         printf("%s\n",trace_event_name);
125885         /* Read a trace event */
125886         posix_trace_getnext_event(trid, &trace_event,
125887             NULL, 0, &returndatasize,&return_value);
125888     }

125889     /* Close the trace stream */
125890     posix_trace_close(trid);
125891     /* Close the trace log */
125892     close(fd);
125893 }

```

125894 Second Example

125895 The complete example includes the two other examples in [Retrieve Information from a Trace Log](#) (on page 3680) and in [Retrieve the List of Trace Event Types Used in a Trace Log](#) (on page 3681). For example, the *maxdatasize* variable is set in [Retrieve the List of Trace Event Types Used in a Trace Log](#) (on page 3681).

```

125899     /* Caution. Error checks omitted */
125900     {
125901         int fd;
125902         trace_id_t trid;
125903         posix_trace_event_info trace_event;
125904         char trace_event_name[TRACE_EVENT_NAME_MAX];
125905         char * data;
125906         size_t maxdatasize=1024, returndatasize;
125907         int return_value;
125908         - - - - -

125909         /* Open an existing trace log */
125910         fd=open("/tmp/tracelog", O_RDONLY);
125911         /* Open a trace stream on the open log */
125912         posix_trace_open( fd, &trid);
125913         /*
125914          * Retrieve information about the trace stream which
125915          * was dumped in this trace log (see example)
125916          */
125917         - - - - -

```

```

125918     /* Allocate a buffer for trace event data */
125919     data=(char *)malloc(maxdatasize);
125920     /*
125921     * Retrieve the list of trace events used in this
125922     * trace log (see example)
125923     */
125924     - - - - -

125925     /* Read and print all trace event names and data out in a loop */
125926     while (1)
125927     {
125928     posix_trace_getnext_event(trid, &trace_event,
125929         data, maxdatasize, &returndatasize,&return_value);
125930         if (return_value != NULL) break;
125931         /*
125932         * Get the name of the trace event type associated
125933         * with trid trace ID
125934         */
125935         posix_trace_eventid_get_name(trid, trace_event.event_id,
125936             trace_event_name);
125937         {
125938         int i;

125939         /* Print the trace event name out */
125940         printf("%s: ", trace_event_name);
125941         /* Print the trace event data out */
125942         for (i=0; i<returndatasize, i++) printf("%02.2X",
125943             (unsigned char)data[i]);
125944         printf("\n");
125945         }
125946     }

125947     /* Close the trace stream */
125948     posix_trace_close(trid);
125949     /* The buffer data is deallocated */
125950     free(data);
125951     /* Now the file can be closed */
125952     close(fd);
125953 }

```

125954 Several Programming Manipulations

125955 The following examples show some typical sets of operations needed in some contexts.

125956 Trace Stream Attribute Manipulation

125957 This example shows the manipulation of a trace stream attribute object in order to change the
125958 default value provided by a previous *posix_trace_attr_init()* call.

```

125959     /* Caution. Error checks omitted */
125960     {
125961         trace_attr_t attr;
125962         size_t logsize=100000;
125963         - - - - -
125964         /* Initialize trace stream attributes */

```

```

125965     posix_trace_attr_init(&attr);
125966     /* Set the trace name in the attributes structure */
125967     posix_trace_attr_setname(&attr, "my_trace");
125968     /* Set the trace full policy */
125969     posix_trace_attr_setstreamfullpolicy(&attr, POSIX_TRACE_LOOP);
125970     /* Set the trace log size */
125971     posix_trace_attr_setlogsize(&attr, logsize);
125972     - - - - -
125973 }

```

125974 **Create a Trace Event Type Set and Change the Trace Event Type Filter**

125975 This example is valid only if the Trace Event Filter option is supported. This example shows the
 125976 manipulation of a trace event type set in order to change the trace event type filter for an
 125977 existing active trace stream, which may be just-created, running, or suspended. Some sets of
 125978 trace event types are well-known, such as the set of trace event types not associated with a
 125979 process, some trace event types are just-built trace event types for this trace stream; one trace
 125980 event type is the predefined trace event error type which is deleted from the trace event type set.

```

125981 /* Caution. Error checks omitted */
125982 {
125983     trace_id_t trid = existing_trace;
125984     trace_event_set_t set;
125985     trace_event_id_t trace_event1, trace_event2;
125986     - - - - -
125987     /* Initialize to an empty set of trace event types */
125988     /* (not strictly required because posix_trace_event_set_fill() */
125989     /* will ignore the prior contents of the event set.) */
125990     posix_trace_eventset_emptyset(&set);
125991     /*
125992     * Fill the set with all system trace events
125993     * not associated with a process
125994     */
125995     posix_trace_eventset_fill(&set, POSIX_TRACE_WOPID_EVENTS);
125996     /*
125997     * Get the trace event type identifier of the known trace event name
125998     * my_first_event for the trid trace stream
125999     */
126000     posix_trace_trid_eventid_open(trid, "my_first_event", &trace_event1);
126001     /* Add the set with this trace event type identifier */
126002     posix_trace_eventset_add_event(trace_event1, &set);
126003     /*
126004     * Get the trace event type identifier of the known trace event name
126005     * my_second_event for the trid trace stream
126006     */
126007     posix_trace_trid_eventid_open(trid, "my_second_event", &trace_event2);
126008     /* Add the set with this trace event type identifier */
126009     posix_trace_eventset_add_event(trace_event2, &set);
126010     - - - - -
126011     /* Delete the system trace event POSIX_TRACE_ERROR from the set */
126012     posix_trace_eventset_del_event(POSIX_TRACE_ERROR, &set);
126013     - - - - -

```

```

126014      /* Modify the trace stream filter making it equal to the new set */
126015      posix_trace_set_filter(trid, &set, POSIX_TRACE_SET_EVENTSET);
126016      - - - - -
126017      /*
126018      * Now trace_event1, trace_event2, and all system trace event types
126019      * not associated with a process, except for the POSIX_TRACE_ERROR
126020      * system trace event type, are filtered out of (not recorded in) the
126021      * existing trace stream.
126022      */
126023  }

```

126024 **Retrieve Information from a Trace Log**

126025 This example shows how to extract information from a trace log, the dump of a trace stream.
 126026 This code:

```

126027      Asks if the trace stream has lost trace events

126028      Extracts the information about the version of the trace subsystem which generated this
126029      trace log

126030      Retrieves the maximum size of trace event data; this may be used to dynamically allocate
126031      an array for extracting trace event data from the trace log without overflow

126032  /* Caution. Error checks omitted */
126033  {
126034      struct posix_trace_status_info statusinfo;
126035      trace_attr_t attr;
126036      trace_id_t trid = existing_trace;
126037      size_t maxdatasize;
126038      char genversion[TRACE_NAME_MAX];
126039      - - - - -
126040      /* Get the trace stream status */
126041      posix_trace_get_status(trid, &statusinfo);
126042      /* Detect an overrun condition */
126043      if (statusinfo.posix_stream_overrun_status == POSIX_TRACE_OVERRUN)
126044          printf("trace events have been lost\n");

126045      /* Get attributes from the trid trace stream */
126046      posix_trace_get_attr(trid, &attr);
126047      /* Get the trace generation version from the attributes */
126048      posix_trace_attr_getgenversion(&attr, genversion);
126049      /* Print the trace generation version out */
126050      printf("Information about Trace Generator:%s\n",genversion);

126051      /* Get the trace event max data size from the attributes */
126052      posix_trace_attr_getmaxdatasize(&attr, &maxdatasize);
126053      /* Print the trace event max data size out */
126054      printf("Maximum size of associated data:%d\n",maxdatasize);
126055      /* Destroy the trace stream attributes */
126056      posix_trace_attr_destroy(&attr);
126057  }

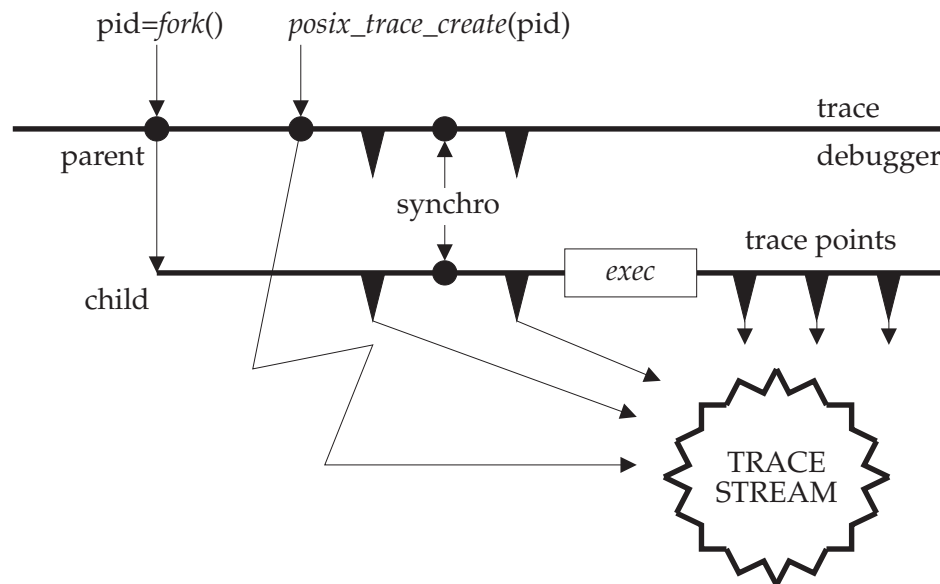
```

126058 Retrieve the List of Trace Event Types Used in a Trace Log

126059 This example shows the retrieval of a trace stream's trace event type list. This operation may be
126060 very useful if you are interested only in tracking the type of trace events in a trace log.

```
126061 /* Caution. Error checks omitted */
126062 {
126063     trace_id_t trid = existing_trace;
126064     trace_event_id_t event_id;
126065     char event_name[TRACE_EVENT_NAME_MAX];
126066     int return_value;
126067     - - - - -
126068     /*
126069      * In a loop print all existing trace event names out
126070      * for the trid trace stream
126071      */
126072     while (1)
126073     {
126074         posix_trace_eventtypelist_getnext_id(trid, &event_id
126075             &return_value);
126076         if (return_value != NULL) break;
126077         /*
126078          * Get the name of the trace event associated
126079          * with trid trace ID
126080          */
126081         posix_trace_eventid_get_name(trid, event_id, event_name);
126082         /* Print the name out */
126083         printf("%s\n", event_name);
126084     }
126085 }
```


126086 B.2.11.4 Rationale on Trace for Debugging



126087

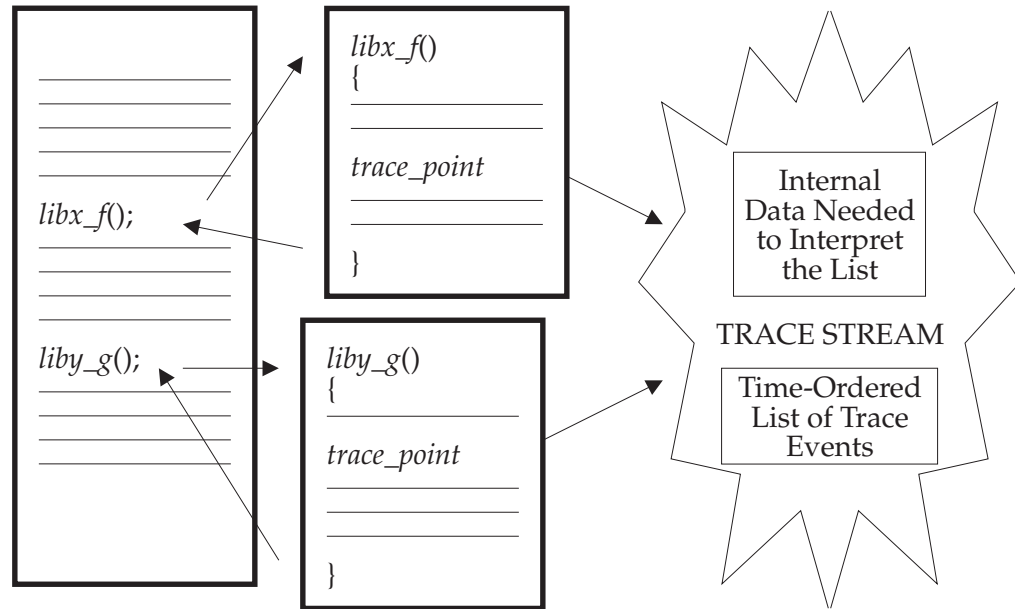
Figure B-5 Trace Another Process

126088 Among the different possibilities offered by the trace interface defined in POSIX.1-2017, the
 126089 debugging of an application is the most interesting one. Typical operations in the controlling
 126090 debugger process are to filter trace event types, to get trace events from the trace stream, to stop
 126091 the trace stream when the debugged process is executing uninteresting code, to start the trace
 126092 stream when some interesting point is reached, and so on. The interface defined in POSIX.1-2017
 126093 should define all the necessary base functions to allow this dynamic debug handling.

126094 **Figure B-5** shows an example in which the trace stream is created after the call to the *fork()*
 126095 function. If the user does not want to lose trace events, some synchronization mechanism
 126096 (represented in the figure) may be needed before calling the *exec()* function, to give the parent a
 126097 chance to create the trace stream before the child begins the execution of its trace points.

126098 B.2.11.5 Rationale on Trace Event Type Name Space

126099 At first, the working group was in favor of the representation of a trace event type by an integer
 126100 (*event_name*). It seems that existing practice shows the weakness of such a representation. The
 126101 collision of trace event types is the main problem that cannot be simply resolved using this sort
 126102 of representation. Suppose, for example, that a third party designs an instrumented library. The
 126103 user does not have the source of this library and wants to trace his application which uses in
 126104 some part the third-party library. There is no means for him to know what are the trace event
 126105 types used in the instrumented library so he has some chance of duplicating some of them and
 126106 thus to obtain a contaminated tracing of his application.



126107

Figure B-6 Trace Name Space Overview: With Third-Party Library

126108 There are requirements to allow program images containing pieces from various vendors to be
 126109 traced without also requiring those of any other vendors to coordinate their uses of the trace
 126110 facility, and especially the naming of their various trace event types and trace point IDs. The
 126111 chosen solution is to provide a very large name space, large enough so that the individual
 126112 vendors can give their trace types and tracepoint IDs sufficiently long and descriptive names
 126113 making the occurrence of collisions quite unlikely. The probability of collision is thus made
 126114 sufficiently low so that the problem may, as a practical matter, be ignored. By requirement, the
 126115 consequence of collisions will be a slight ambiguity in the trace streams; tracing will continue in
 126116 spite of collisions and ambiguities. “The show must go on”. The *posix_prog_address* member of
 126117 the **posix_trace_event_info** structure is used to allow trace streams to be unambiguously
 126118 interpreted, despite the fact that trace event types and trace event names need not be unique.

126119 The *posix_trace_eventid_open()* function is required to allow the instrumented third-party library
 126120 to get a valid trace event type identifier for its trace event names. This operation is, somehow,
 126121 an allocation, and the group was aware of proposing some deallocation mechanism which the
 126122 instrumented application could use to recover the resources used by a trace event type identifier.
 126123 This would have given the instrumented application the benefit of being capable of reusing a
 126124 possible minimum set of trace event type identifiers, but also the inconvenience to have,
 126125 possibly in the same trace stream, one trace event type identifier identifying two different trace
 126126 event types. After some discussions the group decided to not define such a function which
 126127 would make this API thicker for little benefit, the user having always the possibility of adding
 126128 identification information in the *data* member of the trace event structure.

126129 The set of the trace event type identifiers the controlling process wants to filter out is initialized
 126130 in the trace mechanism using the function *posix_trace_set_filter()*, setting the arguments
 126131 according to the definitions explained in *posix_trace_set_filter()*. This operation can be done
 126132 statically (when the trace is in the STOPPED state) or dynamically (when the trace is in the
 126133 STARTED state). The preparation of the filter is normally done using the function defined in
 126134 *posix_trace_eventtypelist_getnext_id()* and eventually the function
 126135 *posix_trace_eventtypelist_rewind()* in order to know (before the recording) the list of the potential

126136 set of trace event types that can be recorded. In the case of an active trace stream, this list may
 126137 not be exhaustive. Actually, the target process may not have yet called the function
 126138 *posix_trace_eventid_open()*. But it is a common practice, for a controlling process, to prepare the
 126139 filtering of a future trace stream before its start. Therefore the user must have a way to get the
 126140 trace event type identifier corresponding to a well-known trace event name before its future
 126141 association by the pre-cited function. This is done by calling the *posix_trace_trid_eventid_open()*
 126142 function, given the trace stream identifier and the trace name, and described hereafter. Because
 126143 this trace event type identifier is associated with a trace stream identifier, where a unique
 126144 process has initialized two or more traces, the implementation is expected to return the same
 126145 trace event type identifier for successive calls to *posix_trace_trid_eventid_open()* with different
 126146 trace stream identifiers. The *posix_trace_eventid_get_name()* function is used by the controller
 126147 process to identify, by the name, the trace event type returned by a call to the
 126148 *posix_trace_eventtypelist_getnext_id()* function.

126149 Afterwards, the set of trace event types is constructed using the functions defined in
 126150 *posix_trace_eventset_empty()*, *posix_trace_eventset_fill()*, *posix_trace_eventset_add()*, and
 126151 *posix_trace_eventset_del()*.

126152 A set of functions is provided devoted to the manipulation of the trace event type identifier and
 126153 names for an active trace stream. All these functions require the trace stream identifier argument
 126154 as the first parameter. The opacity of the trace event type identifier implies that the user cannot
 126155 associate directly its well-known trace event name with the system-associated trace event type
 126156 identifier.

126157 The *posix_trace_trid_eventid_open()* function allows the application to get the system trace event
 126158 type identifier back from the system, given its well-known trace event name. This function is
 126159 useful only when a controlling process needs to specify specific events to be filtered.

126160 The *posix_trace_eventid_get_name()* function allows the application to obtain a trace event name
 126161 given its trace event type identifier. One possible use of this function is to identify the type of a
 126162 trace event retrieved from the trace stream, and print it. The easiest way to implement this
 126163 requirement, is to use a single trace event type map for all the processes whose maps are
 126164 required to be identical. A more difficult way is to attempt to keep multiple maps identical at
 126165 every call to *posix_trace_eventid_open()* and *posix_trace_trid_eventid_open()*.

126166 B.2.11.6 Rationale on Trace Events Type Filtering

126167 The most basic rationale for runtime and pre-registration filtering (selection/rejection) of trace
 126168 event types is to prevent choking of the trace collection facility, and/or overloading of the
 126169 computer system. Any worthwhile trace facility can bring even the largest computer to its
 126170 knees. Otherwise, everything would be recorded and filtered after the fact; it would be much
 126171 simpler, but impractical.

126172 To achieve debugging, measurement, or whatever the purpose of tracing, the filtering of trace
 126173 event types is an important part of trace analysis. Due to the fact that the trace events are put
 126174 into a trace stream and probably logged afterwards into a file, different levels of filtering ‡that
 126175 is, rejection of trace event types—are possible.

126176 **Filtering of Trace Event Types Before Tracing**

126177 This function, represented by the `posix_trace_set_filter()` function in POSIX.1-2017 (see
 126178 `posix_trace_set_filter()`), selects, before or during tracing, the set of trace event types to be filtered
 126179 out. It should be possible also (as OSF suggested in their ETAP trace specifications) to select the
 126180 kernel trace event types to be traced in a system-wide fashion. These two functionalities are
 126181 called the pre-filtering of trace event types.

126182 The restriction on the actual type used for the `trace_event_set_t` type is intended to guarantee
 126183 that these objects can always be assigned, have their address taken, and be passed by value as
 126184 parameters. It is not intended that this type be a structure including pointers to other data
 126185 structures, as that could impact the portability of applications performing such operations. A
 126186 reasonable implementation could be a structure containing an array of integer types.

126187 **Filtering of Trace Event Types at Runtime**

126188 It is possible to build this functionality using the `posix_trace_set_filter()` function. A privileged
 126189 process or a privileged thread can get trace events from the trace stream of another process or
 126190 thread, and thus specify the type of trace events to record into a file, using implementation-
 126191 defined methods and interfaces. This functionality, called inline filtering of trace event types, is
 126192 used for runtime analysis of trace streams.

126193 **Post-Mortem Filtering of Trace Event Types**

126194 The word “post-mortem” is used here to indicate that some unanticipated situation occurs
 126195 during execution that does not permit a pre or inline filtering of trace events and that it is
 126196 necessary to record all trace event types to have a chance to discover the problem afterwards.
 126197 When the program stops, all the trace events recorded previously can be analyzed in order to
 126198 find the solution. This functionality could be named the post-filtering of trace event types.

126199 **Discussions about Trace Event Type Filtering**

126200 After long discussions with the parties involved in the process of defining the trace interface, it
 126201 seems that the sensitivity to the filtering problem is different, but everybody agrees that the level
 126202 of the overhead introduced during the tracing operation depends on the filtering method
 126203 elected. If the time that it takes the trace event to be recorded can be neglected, the overhead
 126204 introduced by the filtering process can be classified as follows:

126205 Pre-filtering System and process/thread-level overhead

126206 Inline-filtering Process/thread-level overhead

126207 Post-filtering No overhead; done offline

126208 The pre-filtering could be named “critical realtime” filtering in the sense that the filtering of
 126209 trace event type is manageable at the user level so the user can lower to a minimum the filtering
 126210 overhead at some user selected level of priority for the inline filtering, or delay the filtering to
 126211 after execution for the post-filtering. The counterpart of this solution is that the size of the trace
 126212 stream must be sufficient to record all the trace events. The advantage of the pre-filtering is that
 126213 the utilization of the trace stream is optimized.

126214 Only pre-filtering is defined by POSIX.1-2017. However, great care must be taken in specifying
 126215 pre-filtering, so that it does not impose unacceptable overhead. Moreover, it is necessary to
 126216 isolate all the functionality relative to the pre-filtering.

126217 The result of this rationale is to define a new option, the Trace Event Filter option, not
 126218 necessarily implemented in small realtime systems, where system overhead is minimized to the
 126219 extent possible.

126220 B.2.11.7 Tracing, pthread API

126221 The objective to be able to control tracing for individual threads may be in conflict with the
 126222 efficiency expected in threads with a *contentionscope* attribute of PTHREAD_SCOPE_PROCESS.
 126223 For these threads, context switches from one thread that has tracing enabled to another thread
 126224 that has tracing disabled may require a kernel call to inform the kernel whether it has to trace
 126225 system events executed by that thread or not. For this reason, it was proposed that the ability to
 126226 enable or disable tracing for PTHREAD_SCOPE_PROCESS threads be made optional, through
 126227 the introduction of a Trace Scope Process option. A trace implementation which did not
 126228 implement the Trace Scope Process option would not honor the tracing-state attribute of a thread
 126229 with PTHREAD_SCOPE_PROCESS; it would, however, honor the tracing-state attribute of a
 126230 thread with PTHREAD_SCOPE_SYSTEM. This proposal was rejected as:

- 126231 1. Removing desired functionality (per-thread trace control)
- 126232 2. Introducing counter-intuitive behavior for the tracing-state attribute
- 126233 3. Mixing logically orthogonal ideas (thread scheduling and thread tracing)
- 126234 [Objective 4]

126235 Finally, to solve this complex issue, this API does not provide *pthread_gettracingstate()*,
 126236 *pthread_settracingstate()*, *pthread_attr_gettracingstate()*, and *pthread_attr_settracingstate()*
 126237 interfaces. These interfaces force the thread implementation to add to the weight of the thread
 126238 and cause a revision of the threads libraries, just to support tracing. Worse yet,
 126239 *posix_trace_event()* must always test this per-thread variable even in the common case where it is
 126240 not used at all. Per-thread tracing is easy to implement using existing interfaces where
 126241 necessary; see the following example.

126242 **Example**

```

126243 /* Caution. Error checks omitted */
126244 static pthread_key_t my_key;
126245 static trace_event_id_t my_event_id;
126246 static pthread_once_t my_once = PTHREAD_ONCE_INIT;

126247 void my_init(void)
126248 {
126249     (void) pthread_key_create(&my_key, NULL);
126250     (void) posix_trace_eventid_open("my", &my_event_id);
126251 }

126252 int get_trace_flag(void)
126253 {
126254     pthread_once(&my_once, my_init);
126255     return (pthread_getspecific(my_key) != NULL);
126256 }

126257 void set_trace_flag(int f)
126258 {
126259     pthread_once(&my_once, my_init);
126260     pthread_setspecific(my_key, f? &my_event_id: NULL);
126261 }

126262 fn()
126263 {
126264     if (get_trace_flag())
126265         posix_trace_event(my_event_id, ...)
126266 }
```

- 126267 The above example does not implement third-party state setting.
- 126268 Lastly, per-thread tracing works poorly for threads with PTHREAD_SCOPE_PROCESS
126269 contention scope. These “library” threads have minimal interaction with the kernel and would
126270 have to explicitly set the attributes whenever they are context switched to a new kernel thread in
126271 order to trace system events. Such state was explicitly avoided in POSIX threads to keep
126272 PTHREAD_SCOPE_PROCESS threads lightweight.
- 126273 The reason that keeping PTHREAD_SCOPE_PROCESS threads lightweight is important is that
126274 such threads can be used not just for simple multi-processors but also for co-routine style
126275 programming (such as discrete event simulation) without inventing a new threads paradigm.
126276 Adding extra runtime cost to thread context switches will make using POSIX threads less
126277 attractive in these situations.
- 126278 *B.2.11.8 Rationale on Triggering*
- 126279 The ability to start or stop tracing based on the occurrence of specific trace event types has been
126280 proposed as a parallel to similar functionality appearing in logic analyzers. Such triggering, in
126281 order to be very useful, should be based not only on the trace event type, but on trace event-
126282 specific data, including tests of user-specified fields for matching or threshold values.
- 126283 Such a facility is unnecessary where the buffering of the stream is not a constraint, since such
126284 checks can be performed offline during post-mortem analysis.
- 126285 For example, a large system could incorporate a daemon utility to collect the trace records from
126286 memory buffers and spool them to secondary storage for later analysis. In the instances where
126287 resources are truly limited, such as embedded applications, the application incorporation of
126288 application code to test the circumstances of a trace event and call the trace point only if needed
126289 is usually straightforward.
- 126290 For performance reasons, the *posix_trace_event()* function should be implemented using a macro,
126291 so if the trace is inactive, the trace event point calls are latent code and must cost no more than a
126292 scalar test.
- 126293 The API proposed in POSIX.1-2017 does not include any triggering functionality.
- 126294 *B.2.11.9 Rationale on Timestamp Clock*
- 126295 It has been suggested that the tracing mechanism should include the possibility of specifying the
126296 clock to be used in timestamping the trace events. When application trace events must be
126297 correlated to remote trace events, such a facility could provide a global time reference not
126298 available from a local clock. Further, the application may be driven by timers based on a clock
126299 different from that used for the timestamp, and the correlation of the trace to those untraced
126300 timer activities could be an important part of the analysis of the application.
- 126301 However, the tracing mechanism needs to be fast and just the provision of such an option can
126302 materially affect its performance. Leaving aside the performance costs of reading some clocks,
126303 this notion is also ill-defined when kernel trace events are to be traced by two applications
126304 making use of different tracing clocks. This can even happen within a single application where
126305 different parts of the application are served by different clocks. Another complication can occur
126306 when a clock is maintained strictly at the user level and is unavailable at the kernel level.
- 126307 It is felt that the benefits of a selectable trace clock do not match its costs. Applications that wish
126308 to correlate clocks other than the default tracing clock can include trace events with sample
126309 values of those other clocks, allowing correlation of timestamps from the various independent
126310 clocks. In any case, such a technique would be required when applications are sensitive to

126311 multiple clocks.

126312 *B.2.11.10 Rationale on Different Overrun Conditions*

126313 The analysis of the dynamic behavior of the trace mechanism shows that different overrun
126314 conditions may occur. The API must provide a means to manage such conditions in a portable
126315 way.

126316 **Overrun in Trace Streams Initialized with POSIX_TRACE_LOOP Policy**

126317 In this case, the user of the trace mechanism is interested in using the trace stream with
126318 POSIX_TRACE_LOOP policy to record trace events continuously, but ideally without losing any
126319 trace events. The online analyzer process must get the trace events at a mean speed equivalent to
126320 the recording speed. Should the trace stream become full, a trace stream overrun occurs. This
126321 condition is detected by getting the status of the active trace stream (function
126322 *posix_trace_get_status()*) and looking at the member *posix_stream_overrun_status* of the read
126323 **posix_stream_status** structure. In addition, two predefined trace event types are defined:

- 126324 1. The beginning of a trace overflow, to locate the beginning of an overflow when reading a
126325 trace stream
- 126326 2. The end of a trace overflow, to locate the end of an overflow, when reading a trace stream

126327 As a timestamp is associated with these predefined trace events, it is possible to know the
126328 duration of the overflow.

126329 **Overrun in Dumping Trace Streams into Trace Logs**

126330 The user lets the trace mechanism dump the trace stream initialized with
126331 POSIX_TRACE_FLUSH policy automatically into a trace log. If the dump operation is slower
126332 than the recording of trace events, the trace stream can overrun. This condition is detected by
126333 getting the status of the active trace stream (the *posix_trace_get_status()* function) and looking at
126334 the member *posix_stream_overrun_status* of the read **posix_stream_status** structure. This overrun
126335 indicates that the trace mechanism is not able to operate in this mode at this speed. It is the
126336 responsibility of the user to modify one of the trace parameters (the stream size or the trace
126337 event type filter, for instance) to avoid such overrun conditions, if overruns are to be prevented.
126338 The same already predefined trace event types (see [Overrun in Trace Streams Initialized with
126339 POSIX_TRACE_LOOP Policy](#)) are used to detect and to know the duration of an overflow.

126340 **Reading an Active Trace Stream**

126341 Although this trace API allows one to read an active trace stream with log while it is tracing, this
126342 feature can lead to false overflow origin interpretation: the trace log or the reader of the trace
126343 stream. Reading from an active trace stream with log is thus non-portable, and has been left
126344 unspecified.

126345 **B.2.12 Data Types**126346 *B.2.12.1 Defined Types*

126347 The requirement that additional types defined in this section end in ``_t'' was prompted by the
 126348 problem of name space pollution. It is difficult to define a type (where that type is not one
 126349 defined by POSIX.1-2017) in one header file and use it in another without adding symbols to the
 126350 name space of the program. To allow implementors to provide their own types, all conforming
 126351 applications are required to avoid symbols ending in ``_t'', which permits the implementor to
 126352 provide additional types. Because a major use of types is in the definition of structure members,
 126353 which can (and in many cases must) be added to the structures defined in POSIX.1-2017, the
 126354 need for additional types is compelling.

126355 The types, such as **ushort** and **ulong**, which are in common usage, are not defined in
 126356 POSIX.1-2017 (although **ushort_t** would be permitted as an extension). They can be added to
 126357 **<sys/types.h>** using a feature test macro (see [Section B.2.2.1](#), on page 3566). A suggested symbol
 126358 for these is **_SYSIII**. Similarly, the types like **u_short** would probably be best controlled by **_BSD**.

126359 Some of these symbols may appear in other headers; see [Section B.2.2.2](#) (on page 3567).

126360 **dev_t** This type may be made large enough to accommodate host-locality considerations
 126361 of networked systems.

126362 This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic
 126363 (such as a structure) and provided a *samefile()* function for comparison.

126364 **gid_t** Some implementations had separated **gid_t** from **uid_t** before POSIX.1 was
 126365 completed. It would be difficult for them to coalesce them when it was
 126366 unnecessary. Additionally, it is quite possible that user IDs might be different from
 126367 group IDs because the user ID might wish to span a heterogeneous network,
 126368 where the group ID might not.

126369 For current implementations, the cost of having a separate **gid_t** will be only
 126370 lexical.

126371 **mode_t** This type was chosen so that implementations could choose the appropriate
 126372 integer type, and for compatibility with the ISO C standard. 4.3 BSD uses
 126373 **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the
 126374 low-order sixteen bits are significant.

126375 **nlink_t** This type was introduced in place of **short** for *st_nlink* (see the **<sys/stat.h>** header)
 126376 in response to an objection that **short** was too small.

126377 **off_t** This type is used to represent a file offset or file size. On systems supporting large
 126378 files, **off_t** is larger than 32 bits in at least one programming environment. Other
 126379 programming environments may use different sizes for **off_t**, for compatibility or
 126380 other reasons.

126381 **pid_t** The inclusion of this symbol was controversial because it is tied to the issue of the
 126382 representation of a process ID as a number. From the point of view of a
 126383 conforming application, process IDs should be ``magic cookies''⁸ that are produced
 126384 by calls such as *fork()*, used by calls such as *waitpid()* or *kill()*, and not otherwise

126385 8. An historical term meaning: ``An opaque object, or token, of determinate size, whose significance is known only to the entity which
 126386 created it. An entity receiving such a token from the generating entity may only make such use of the `cookie' as is defined and permitted
 126387 by the supplying entity.''

126388		analyzed (except that the sign is used as a flag for certain operations).
126389		The concept of a {PID_MAX} value interacted with this in early proposals. Treating process IDs as an opaque type both removes the requirement for {PID_MAX} and allows systems to be more flexible in providing process IDs that span a large range of values, or a small one.
126390		
126391		
126392		
126393		Since the values in uid_t , gid_t , and pid_t will be numbers generally, and potentially both large in magnitude and sparse, applications that are based on arrays of objects of this type are unlikely to be fully portable in any case. Solutions that treat them as magic cookies will be portable.
126394		
126395		
126396		
126397		{CHILD_MAX} precludes the possibility of a “toy implementation”, where there would only be one process.
126398		
126399	ssize_t	This is intended to be a signed analog of size_t . The wording is such that an implementation may either choose to use a longer type or simply to use the signed version of the type that underlies size_t . All functions that return ssize_t (<i>read()</i> and <i>write()</i>) describe as “implementation-defined” the result of an input exceeding {SSIZE_MAX}. It is recognized that some implementations might have ints that are smaller than size_t . A conforming application would be constrained not to perform I/O in pieces larger than {SSIZE_MAX}, but a conforming application using extensions would be able to use the full range if the implementation provided an extended range, while still having a single type-compatible interface.
126400		
126401		
126402		
126403		
126404		
126405		
126406		
126407		
126408		
126409		
126410		
126411		
126412	uid_t	Before the addition of this type, the data types used to represent these values varied throughout early proposals. The <sys/stat.h> header defined these values as type short , the <passwd.h> file (now <pwd.h> and <grp.h>) used an int , and <i>getuid()</i> returned an int . In response to a strong objection to the inconsistent definitions, all the types were switched to uid_t .
126413		
126414		
126415		
126416		In practice, those historical implementations that use varying types of this sort can typedef uid_t to short with no serious consequences.
126417		
126418		
126419		
126420		The problem associated with this change concerns object compatibility after structure size changes. Since most implementations will define uid_t as a short, the only substantive change will be a reduction in the size of the passwd structure. Consequently, implementations with an overriding concern for object compatibility can pad the structure back to its current size. For that reason, this problem was not considered critical enough to warrant the addition of a separate type to POSIX.1.
126421		
126422		
126423		
126424		
126425		
126426		The types uid_t and gid_t are magic cookies. There is no {UID_MAX} defined by POSIX.1, and no structure imposed on uid_t and gid_t other than that they be positive arithmetic types. (In fact, they could be unsigned char .) There is no maximum or minimum specified for the number of distinct user or group IDs.
126427		
126428		
126429		
126430		POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0030 [733] is applied.

126431 B.2.12.2 *The char Type*

126432 POSIX.1-2017 explicitly requires that a **char** type is exactly one byte (8 bits).

126433 B.2.13 Status Information

126434 POSIX.1-2017 does not require all matching WNOWAIT threads (threads in a matching call to *waitid()* with the WNOWAIT flag set) to obtain a child's status information because the status information might be discarded (consumed or replaced) before one of the matching WNOWAIT threads is scheduled. If the status information is not discarded, it will remain available, so all of the matching WNOWAIT threads will (eventually) obtain the status information.

126438 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0031 [690] is applied.

126440 B.2.14 File Descriptor Allocation

126441 Functions such as *pipe()* and *socketpair()* which allocate two file descriptors are permitted to perform the two allocations independently. This means that other threads or signal handlers may perform operations on file descriptors in between the two allocations and this can result in the two file descriptors not having adjacent values or in the second allocation producing a lower value than the first.

126446 POSIX.1-2008, Technical Corrigendum 2, XSH/TC2-2008/0032 [835] is applied.

126447 B.3 System Interfaces

126448 See the RATIONALE sections on the individual reference pages.

126449 B.3.1 System Interfaces Removed in this Version

126450 The following section contains a list of the interfaces removed in POSIX.1-2017, together with advice for application developers on the alternative interfaces that should be used for maximum portability.

126453 B.3.1.1 *bcmp()*

126454 Applications are recommended to use the *memcmp()* function instead of this function.

126455 For maximum portability, it is recommended to replace the function call to *bcmp()* as follows:

```
126456 #define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))
```

126457 B.3.1.2 *bcopy()*

126458 Applications are recommended to use the *memmove()* function instead of this function.

126459 The following are approximately equivalent (note the order of the arguments):

```
126460 bcopy(s1,s2,n) ≈ memmove(s2,s1,n)
```

126461 For maximum portability, it is recommended to replace the function call to *bcopy()* as follows:

```
126462 #define bcopy(b1,b2,len) (void)(memmove((b2), (b1), (len)))
```

- 126463 B.3.1.3 *bsd_signal()*
- 126464 Applications are recommended to use the *sigaction()* function instead of this function.
- 126465 The *bsd_signal()* function was supplied as a migration path for the BSD *signal()* function for
126466 simple applications that installed a single-argument signal handler function.
- 126467 Historically, the *bsd_signal()* function differs from *signal()* in that the SA_RESTART flag is set
126468 and the SA_RESETHAND flag is clear when *bsd_signal()* is used. The state of these flags is not
126469 specified for *signal()*.
- 126470 B.3.1.4 *bzero()*
- 126471 Applications are recommended to use the *memset()* function instead of this function.
- 126472 For maximum portability, it is recommended to replace the function call to *bzero()* as follows:
- 126473

```
#define bzero(b,len) (void)(memset((b), '\0', (len)))
```
- 126474 B.3.1.5 *ecvt(), fcvt(), gcvt()*
- 126475 Applications are recommended to use the *sprintf()* function instead of these functions.
- 126476 The *sprintf()* function is required by ISO C and is thus more portable.
- 126477 B.3.1.6 *ftime()*
- 126478 Applications are recommended to use the *time()* function to determine the current time.
126479 Realtime applications should use *clock_gettime()* to determine the current time.
- 126480 B.3.1.7 *getcontext(), makecontext(), swapcontext()*
- 126481 Due to portability issues with these functions, especially with the manipulation of contexts,
126482 applications are recommended to be rewritten to use POSIX threads.
- 126483 B.3.1.8 *gethostbyaddr(), gethostbyname()*
- 126484 Applications are recommended to use the *getaddrinfo()* and *getnameinfo()* functions instead of
126485 these functions.
- 126486 The *gethostbyaddr()* and *gethostbyname()* functions may return pointers to static data, which may
126487 be overwritten by subsequent calls to any of these functions. The suggested replacements do not
126488 have this problem and are also IPv6-capable.
- 126489 B.3.1.9 *getwd()*
- 126490 Applications are recommended to use the *getcwd()* function to determine the current working
126491 directory.

126492 B.3.1.10 *h_errno*

126493 Applications are recommended not to use this error return code. Previously it was set by the
126494 *gethostbyaddr()* and *gethostbyname()* functions.

126495 B.3.1.11 *index()*

126496 Applications are recommended to use the *strchr()* function instead of this function.

126497 For maximum portability, it is recommended to replace the function call to *index()* as follows:

```
126498 #define index(a,b) strchr((a),(b))
```

126499 B.3.1.12 *makecontext()*

126500 Applications using the *getcontext()*, *makecontext()*, and *swapcontext()* functions should be
126501 rewritten to use POSIX threads.

126502 B.3.1.13 *mktemp()*

126503 Applications are recommended to use the *mkstemp()* function instead of this function.

126504 The *mktemp()* function makes an application vulnerable to possible security problems since
126505 between the time a pathname is created and the file opened, it is possible for some other process
126506 to create a file with the same name. The *mkstemp()* function does not have this vulnerability.

126507 B.3.1.14 *pthread_attr_getstackaddr()*, *pthread_attr_setstackaddr()*

126508 Applications are recommended to use the *pthread_attr_setstack()* and *pthread_attr_getstack()*
126509 functions instead of these functions.

126510 There are a number of ambiguities in the specification of the *stackaddr* attribute that makes
126511 portable use of these interfaces impossible.

126512 B.3.1.15 *rindex()*

126513 Applications are recommended to use the *strrchr()* function instead of this function.

126514 For maximum portability, it is recommended to replace the function call to *rindex()* as follows:

```
126515 #define rindex(a,b) strrchr((a),(b))
```

126516 B.3.1.16 *scalb()*

126517 Applications are recommended to use either *scalbln()*, *scalblnf()*, or *scalblnl()* instead of these
126518 functions.

126519 The behavior for the *scalb()* function was only defined when the *n* argument is an integer, a
126520 NaN, or Inf. The behavior of other values for the *n* argument was unspecified.

126521 B.3.1.17 *ualarm()*

126522 Applications are recommended to use *timer_create()*, *timer_delete()*, *timer_getoverrun()*,
126523 *timer_gettime()*, or *timer_settime()* instead of this function.

126524 B.3.1.18 *usleep()*

126525 Applications are recommended to use the *nanosleep()* function instead of this function.

126526 B.3.1.19 *vfork()*

126527 Applications are recommended to use the *fork()* function instead of this function.

126528 The *vfork()* function was previously under-specified.

126529 B.3.1.20 *wcs wcs()*

126530 Applications are recommended to use the *wcsstr()* function instead of this function.

126531 The *wcsstr()* function is technically equivalent and is portable across all ISO C implementations.

126532 B.3.2 System Interfaces Removed in the Previous Version

126533 The following system interfaces, headers, and external variables were removed in the previous
126534 version of this standard:

126535	<i>advance()</i>	<i>getdtablesize()</i>	<i>re_exec()</i>	<i>ttyslot()</i>	<i>loc1</i>
126536	<i>brk()</i>	<i>getpagesize()</i>	<i>regcmp()</i>	<i>valloc()</i>	<i>__loc1</i>
126537	<i>chroot()</i>	<i>getpass()</i>	<i>regex()</i>	<i>wait3()</i>	<i>loc2</i>
126538	<i>compile()</i>	<i>getw()</i>	<i>sbrk()</i>	<re_comp.h>	<i>locs</i>
126539	<i>cuserid()</i>	<i>putw()</i>	<i>sigstack()</i>	<regex.h>	
126540	<i>gamma()</i>	<i>re_comp()</i>	<i>step()</i>	<varargs.h>	

126541 B.3.3 Examples for Spawn

126542 The following long examples are provided in the Rationale (Informative) volume of
126543 POSIX.1-2017 as a supplement to the reference page for *posix_spawn()*.

126544 Example Library Implementation of Spawn

126545 The *posix_spawn()* or *posix_spawnnp()* functions provide the following:

126546 Simply start a process executing a process image. This is the simplest application for
126547 process creation, and it may cover most executions of *fork()*.

126548 Support I/O redirection, including pipes.

126549 Run the child under a user and group ID in the domain of the parent.

126550 Run the child at any priority in the domain of the parent.

126551 The *posix_spawn()* or *posix_spawnnp()* functions do not cover every possible use of the *fork()*
126552 function, but they do span the common applications: typical use by a shell and a login utility.

126553 The price for an application is that before it calls *posix_spawn()* or *posix_spawnnp()*, the parent
126554 must adjust to a state that *posix_spawn()* or *posix_spawnnp()* can map to the desired state for the

126555 child. Environment changes require the parent to save some of its state and restore it afterwards.
 126556 The effective behavior of a successful invocation of *posix_spawn()* is as if the operation were
 126557 implemented with POSIX operations as follows:

```

126558 #include <sys/types.h>
126559 #include <stdlib.h>
126560 #include <stdio.h>
126561 #include <unistd.h>
126562 #include <sched.h>
126563 #include <fcntl.h>
126564 #include <signal.h>
126565 #include <errno.h>
126566 #include <string.h>
126567 #include <signal.h>

126568 /* #include <spawn.h> */
126569 /*****
126570 /* Things that could be defined in spawn.h */
126571 /*****
126572 typedef struct
126573 {
126574     short posix_attr_flags;
126575     #define POSIX_SPAWN_SETPGROUP      0x1
126576     #define POSIX_SPAWN_SETSIGMASK    0x2
126577     #define POSIX_SPAWN_SETSIGDEF     0x4
126578     #define POSIX_SPAWN_SETSCHEDULER  0x8
126579     #define POSIX_SPAWN_SETSCHEDPARAM 0x10
126580     #define POSIX_SPAWN_RESETPIDS     0x20
126581     pid_t posix_attr_pgroup;
126582     sigset_t posix_attr_sigmask;
126583     sigset_t posix_attr_sigdefault;
126584     int posix_attr_schedpolicy;
126585     struct sched_param posix_attr_schedparam;
126586 } posix_spawnattr_t;

126587 typedef char *posix_spawn_file_actions_t;

126588 int posix_spawn_file_actions_init(
126589     posix_spawn_file_actions_t *file_actions);
126590 int posix_spawn_file_actions_destroy(
126591     posix_spawn_file_actions_t *file_actions);
126592 int posix_spawn_file_actions_addclose(
126593     posix_spawn_file_actions_t *file_actions, int fildes);
126594 int posix_spawn_file_actions_adddup2(
126595     posix_spawn_file_actions_t *file_actions, int fildes,
126596     int newfildes);
126597 int posix_spawn_file_actions_addopen(
126598     posix_spawn_file_actions_t *file_actions, int fildes,
126599     const char *path, int oflag, mode_t mode);
126600 int posix_spawnattr_init(posix_spawnattr_t *attr);
126601 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
126602 int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
126603     short *lags);
126604 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
126605 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,

```

```

126606     pid_t *pgroup);
126607 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
126608 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
126609     int *schedpolicy);
126610 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
126611     int schedpolicy);
126612 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
126613     struct sched_param *schedparam);
126614 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
126615     const struct sched_param *schedparam);
126616 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
126617     sigset_t *sigmask);
126618 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
126619     const sigset_t *sigmask);
126620 int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
126621     sigset_t *sigdefault);
126622 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
126623     const sigset_t *sigdefault);
126624 int posix_spawn(pid_t *pid, const char *path,
126625     const posix_spawn_file_actions_t *file_actions,
126626     const posix_spawnattr_t *attrp, char *const argv[],
126627     char *const envp[]);
126628 int posix_spawnnp(pid_t *pid, const char *file,
126629     const posix_spawn_file_actions_t *file_actions,
126630     const posix_spawnattr_t *attrp, char *const argv[],
126631     char *const envp[]);

126632     /*****
126633     /* Example posix_spawn() library routine */
126634     /*****
126635     int posix_spawn(pid_t *pid,
126636         const char *path,
126637         const posix_spawn_file_actions_t *file_actions,
126638         const posix_spawnattr_t *attrp,
126639         char *const argv[],
126640         char *const envp[])
126641     {
126642         /* Create process */
126643         if ((*pid = fork()) == (pid_t) 0)
126644         {
126645             /* This is the child process */
126646             /* Worry about process group */
126647             if (attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
126648             {
126649                 /* Override inherited process group */
126650                 if (setpgid(0, attrp->posix_attr_pgroup) != 0)
126651                 {
126652                     /* Failed */
126653                     exit(127);
126654                 }
126655             }

126656             /* Worry about thread signal mask */
126657             if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)

```

```

126658     {
126659         /* Set the signal mask (cannot fail) */
126660         sigprocmask(SIG_SETMASK, &attrp->posix_attr_sigmask, NULL);
126661     }

126662     /* Worry about resetting effective user and group IDs */
126663     if (attrp->posix_attr_flags & POSIX_SPAWN_RESETEIDS)
126664     {
126665         /* None of these can fail for this case. */
126666         setuid(getuid());
126667         setgid(getgid());
126668     }

126669     /* Worry about defaulted signals */
126670     if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
126671     {
126672         struct sigaction deflt;
126673         sigset_t all_signals;

126674         int s;

126675         /* Construct default signal action */
126676         deflt.sa_handler = SIG_DFL;
126677         deflt.sa_flags = 0;

126678         /* Construct the set of all signals */
126679         sigfillset(&all_signals);

126680         /* Loop for all signals */
126681         for (s = 0; sigismember(&all_signals, s); s++)
126682         {
126683             /* Signal to be defaulted? */
126684             if (sigismember(&attrp->posix_attr_sigdefault, s))
126685             {
126686                 /* Yes; default this signal */
126687                 if (sigaction(s, &deflt, NULL) == -1)
126688                 {
126689                     /* Failed */
126690                     exit(127);
126691                 }
126692             }
126693         }
126694     }

126695     /* Worry about the fds if they are to be mapped */
126696     if (file_actions != NULL)
126697     {
126698         /* Loop for all actions in object file_actions */
126699         /* (implementation dives beneath abstraction) */
126700         char *p = *file_actions;

126701         while (*p != '\0')
126702         {
126703             if (strncmp(p, "close(", 6) == 0)
126704             {
126705                 int fd;

```



```

126706         if (sscanf(p + 6, "%d", &fd) != 1)
126707         {
126708             exit(127);
126709         }
126710         if (close(fd) == -1)
126711             exit(127);
126712     }
126713     else if (strncmp(p, "dup2(", 5) == 0)
126714     {
126715         int fd, newfd;
126716
126717         if (sscanf(p + 5, "%d,%d", &fd, &newfd) != 2)
126718         {
126719             exit(127);
126720         }
126721         if (dup2(fd, newfd) == -1)
126722             exit(127);
126723     }
126724     else if (strncmp(p, "open(", 5) == 0)
126725     {
126726         int fd, oflag;
126727         mode_t mode;
126728         int tempfd;
126729         char path[1000];    /* Should be dynamic */
126730         char *q;
126731
126732         if (sscanf(p + 5, "%d", &fd) != 1)
126733         {
126734             exit(127);
126735         }
126736         p = strchr(p, ',') + 1;
126737         q = strchr(p, '*');
126738         if (q == NULL)
126739             exit(127);
126740         strncpy(path, p, q - p);
126741         path[q - p] = '\0';
126742         if (sscanf(q + 1, "%o,%o", &oflag, &mode) != 2)
126743         {
126744             exit(127);
126745         }
126746         if (close(fd) == -1)
126747         {
126748             if (errno != EBADF)
126749                 exit(127);
126750         }
126751         tempfd = open(path, oflag, mode);
126752         if (tempfd == -1)
126753             exit(127);
126754         if (tempfd != fd)
126755         {
126756             if (dup2(tempfd, fd) == -1)
126757                 exit(127);
126758         }

```

```

126758             if (close(tempfd) == -1)
126759                 {
126760                     exit(127);
126761                 }
126762             }
126763         }
126764     else
126765     {
126766         exit(127);
126767     }
126768     p = strchr(p, '(') + 1;
126769 }
126770 }
126771 /* Worry about setting new scheduling policy and parameters */
126772 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
126773 {
126774     if (sched_setscheduler(0, attrp->posix_attr_schedpolicy,
126775         &attrp->posix_attr_schedparam) == -1)
126776     {
126777         exit(127);
126778     }
126779 }
126780 /* Worry about setting only new scheduling parameters */
126781 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
126782 {
126783     if (sched_setparam(0, &attrp->posix_attr_schedparam) == -1)
126784     {
126785         exit(127);
126786     }
126787 }
126788 /* Now execute the program at path */
126789 /* Any fd that still has FD_CLOEXEC set will be closed */
126790 execve(path, argv, envp);
126791 exit(127);          /* exec failed */
126792 }
126793 else
126794 {
126795     /* This is the parent (calling) process */
126796     if (*pid == (pid_t) - 1)
126797         return errno;
126798     return 0;
126799 }
126800 }
126801 /******
126802 /* Here is a crude but effective implementation of the */
126803 /* file action object operators which store actions as */
126804 /* concatenated token-separated strings.          */
126805 /******
126806 /* Create object with no actions. */
126807 int posix_spawn_file_actions_init(
126808     posix_spawn_file_actions_t *file_actions)

```

```

126809     {
126810         *file_actions = malloc(sizeof(char));
126811         if (*file_actions == NULL)
126812             return ENOMEM;
126813         strcpy(*file_actions, "");
126814         return 0;
126815     }

126816     /* Free object storage and make invalid. */
126817     int posix_spawn_file_actions_destroy(
126818         posix_spawn_file_actions_t *file_actions)
126819     {
126820         free(*file_actions);
126821         *file_actions = NULL;
126822         return 0;
126823     }

126824     /* Add a new action string to object. */
126825     static int add_to_file_actions(
126826         posix_spawn_file_actions_t *file_actions, char *new_action)
126827     {
126828         *file_actions = realloc
126829             (*file_actions, strlen(*file_actions) + strlen(new_action) + 1);
126830         if (*file_actions == NULL)
126831             return ENOMEM;
126832         strcat(*file_actions, new_action);
126833         return 0;
126834     }

126835     /* Add a close action to object. */
126836     int posix_spawn_file_actions_addclose(
126837         posix_spawn_file_actions_t *file_actions, int fildes)
126838     {
126839         char temp[100];

126840         sprintf(temp, "close(%d)", fildes);
126841         return add_to_file_actions(file_actions, temp);
126842     }

126843     /* Add a dup2 action to object. */
126844     int posix_spawn_file_actions_adddup2(
126845         posix_spawn_file_actions_t *file_actions, int fildes,
126846         int newfildes)
126847     {
126848         char temp[100];

126849         sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
126850         return add_to_file_actions(file_actions, temp);
126851     }

126852     /* Add an open action to object. */
126853     int posix_spawn_file_actions_addopen(
126854         posix_spawn_file_actions_t *file_actions, int fildes,
126855         const char *path, int oflag, mode_t mode)
126856     {
126857         char temp[100];

```

```

126858     sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
126859     return add_to_file_actions(file_actions, temp);
126860 }

126861 /*****
126862  /* Here is a crude but effective implementation of the */
126863  /* spawn attributes object functions which manipulate */
126864  /* the individual attributes. */
126865  *****/
126866  /* Initialize object with default values. */
126867  int posix_spawnattr_init(posix_spawnattr_t *attr)
126868  {
126869     attr->posix_attr_flags = 0;
126870     attr->posix_attr_pgroup = 0;
126871     /* Default value of signal mask is the parent's signal mask; */
126872     /* other values are also allowed */
126873     sigprocmask(0, NULL, &attr->posix_attr_sigmask);
126874     sigemptyset(&attr->posix_attr_sigdefault);
126875     /* Default values of scheduling attr inherited from the parent; */
126876     /* other values are also allowed */
126877     attr->posix_attr_schedpolicy = sched_getscheduler(0);
126878     sched_getparam(0, &attr->posix_attr_schedparam);
126879     return 0;
126880 }

126881  int posix_spawnattr_destroy(posix_spawnattr_t *attr)
126882  {
126883     /* No action needed */
126884     return 0;
126885 }

126886  int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
126887     short *flags)
126888  {
126889     *flags = attr->posix_attr_flags;
126890     return 0;
126891 }

126892  int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
126893  {
126894     attr->posix_attr_flags = flags;
126895     return 0;
126896 }

126897  int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
126898     pid_t *pgroup)
126899  {
126900     *pgroup = attr->posix_attr_pgroup;
126901     return 0;
126902 }

126903  int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
126904  {
126905     attr->posix_attr_pgroup = pgroup;
126906     return 0;
126907 }

```

```
126908     int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
126909         int *schedpolicy)
126910     {
126911         *schedpolicy = attr->posix_attr_schedpolicy;
126912         return 0;
126913     }

126914     int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
126915         int schedpolicy)
126916     {
126917         attr->posix_attr_schedpolicy = schedpolicy;
126918         return 0;
126919     }

126920     int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
126921         struct sched_param *schedparam)
126922     {
126923         *schedparam = attr->posix_attr_schedparam;
126924         return 0;
126925     }

126926     int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
126927         const struct sched_param *schedparam)
126928     {
126929         attr->posix_attr_schedparam = *schedparam;
126930         return 0;
126931     }

126932     int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
126933         sigset_t *sigmask)
126934     {
126935         *sigmask = attr->posix_attr_sigmask;
126936         return 0;
126937     }

126938     int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
126939         const sigset_t *sigmask)
126940     {
126941         attr->posix_attr_sigmask = *sigmask;
126942         return 0;
126943     }

126944     int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
126945         sigset_t *sigdefault)
126946     {
126947         *sigdefault = attr->posix_attr_sigdefault;
126948         return 0;
126949     }

126950     int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
126951         const sigset_t *sigdefault)
126952     {
126953         attr->posix_attr_sigdefault = *sigdefault;
126954         return 0;
126955     }
```

126956 **I/O Redirection with Spawn**

126957 I/O redirection with *posix_spawn()* or *posix_spawnnp()* is accomplished by crafting a *file_actions*
126958 argument to effect the desired redirection. Such a redirection follows the general outline of the
126959 following example:


```
126960 /* To redirect new standard output (fd 1) to a file, */  
126961 /* and redirect new standard input (fd 0) from my fd socket_pair[1], */  
126962 /* and close my fd socket_pair[0] in the new process. */  
126963 posix_spawn_file_actions_t file_actions;  
126964 posix_spawn_file_actions_init(&file_actions);  
126965 posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);  
126966 posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);  
126967 posix_spawn_file_actions_close(&file_actions, socket_pair[0]);  
126968 posix_spawn_file_actions_close(&file_actions, socket_pair[1]);  
126969 posix_spawn(..., &file_actions, ...);  
126970 posix_spawn_file_actions_destroy(&file_actions);
```

126971 **Spawning a Process Under a New User ID**

126972 Spawning a process under a new user ID follows the outline shown in the following example:

```
126973 Save = getuid();  
126974 setuid(newid);  
126975 posix_spawn(...);  
126976 setuid(Save);
```


126977

 *Rationale (Informative)*

126978

Part C:

126979

Shell and Utilities

126980

The Open Group

126981

The Institute of Electrical and Electronics Engineers, Inc.

Rationale for Shell and Utilities

126984 C.1 Introduction

126985 C.1.1 Change History

126986 The change history is provided as an informative section, to track changes from earlier versions
126987 of this standard.

126988 The following sections describe changes made to the Shell and Utilities volume of POSIX.1-2017
126989 since Issue 6 of the base document. The CHANGE HISTORY section for each utility describes
126990 technical changes made to that utility from Issue 5. Changes between earlier versions of the base
126991 document and Issue 5 are not included.

126992 Changes from Issue 6 to Issue 7 (POSIX.1-2008)

126993 The following list summarizes the major changes that were made in the Shell and Utilities
126994 volume of POSIX.1-2017 from Issue 6 to Issue 7:

126995 Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses
126996 to ISO/IEC defect reports against ISO/IEC 9945 are applied.

126997 The Open Group corrigenda and resolutions are applied.

126998 Features, marked legacy or obsolescent in the base document, have been considered for
126999 removal in this version.

127000 A review of the use of fixed pathnames within the standard has been undertaken; for
127001 example, the *at*, *batch*, and *crontab* utilities previously had a requirement for use of the
127002 directory */usr/lib/cron*.

127003 The options within the standard have been revised.

127004 ‡ The Batch Environment Services and Utilities option is marked obsolescent.

127005 ‡ The UUCP utilities option is added.

127006 ‡ The User Portability Utilities option is revised so that only the *bg*, *ex*, *fc*, *fg*, *jobs*, *more*,
127007 *talk*, and *vi* utilities are included, the rest being moved to the Base.

127008 **New Features in Issue 7**

127009 There are no new utilities in Issue 7.

127010 **C.1.2 Relationship to Other Documents**127011 *C.1.2.1 System Interfaces*

127012 It has been pointed out that the Shell and Utilities volume of POSIX.1-2017 assumes that a great
 127013 deal of functionality from the System Interfaces volume of POSIX.1-2017 is present, but never
 127014 states exactly how much (and strictly does not need to since both are mandated on a conforming
 127015 system). This section is an attempt to clarify the assumptions.

127016 **File Read, Write, and Creation**

127017 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/2 is applied, updating Table 1-1.

127018 **File Removal**

127019 This is intended to be a summary of the *unlink()* and *rmdir()* requirements. Note that it is
 127020 possible using the *unlink()* function for item 4. to occur.

127021 *C.1.2.2 Concepts Derived from the ISO C Standard*

127022 This section was introduced to address the issue that there was insufficient detail presented by
 127023 such utilities as *awk* or *sh* about their procedural control statements and their methods of
 127024 performing arithmetic functions.

127025 The ISO C standard was selected as a model because most historical implementations of the
 127026 standard utilities were written in C. Thus, it was more likely that they would act in the desired
 127027 manner without modification.

127028 Using the ISO C standard is primarily a notational convenience so that the many procedural
 127029 languages in the Shell and Utilities volume of POSIX.1-2017 would not have to be rigorously
 127030 described in every aspect. Its selection does not require that the standard utilities be written in
 127031 Standard C; they could be written in Common Usage C, Ada, Pascal, assembler language, or
 127032 anything else.

127033 The sizes of the various numeric values refer to C-language data types that are allowed to be
 127034 different sizes by the ISO C standard. Thus, like a C-language application, a shell application
 127035 cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the
 127036 ISO C standard, such as {LONG_MAX} for a **long** type.

127037 The behavior on overflow is undefined for ISO C standard arithmetic. Therefore, the standard
 127038 utilities can use “bignum” representation for integers so that there is no fixed maximum unless
 127039 otherwise stated in the utility description. Similarly, standard utilities can use infinite-precision
 127040 representations for floating-point arithmetic, as long as these representations exceed the ISO C
 127041 standard requirements.

127042 This section addresses only the issue of semantics; it is not intended to specify syntax. For
 127043 example, the ISO C standard requires that 0L be recognized as an integer constant equal to zero,
 127044 but utilities such as *awk* and *sh* are not required to recognize 0L (though they are allowed to, as
 127045 an extension).

127046 The ISO C standard requires that a C compiler must issue a diagnostic for constants that are too
 127047 large to represent. Most standard utilities are not required to issue these diagnostics; for
 127048 example, the command:

```
127049 diff -C 2147483648 file1 file2
```

127050 has undefined behavior, and the *diff* utility is not required to issue a diagnostic even if the
 127051 number 2 147 483 648 cannot be represented.

127052 C.1.3 Utility Limits

127053 This section grew out of an idea that originated with the original POSIX.1, in the tables of system
 127054 limits for the *sysconf()* and *pathconf()* functions. The idea being that a conforming application
 127055 can be written to use the most restrictive values that a minimal system can provide, but it should
 127056 not have to. The values provided represent compromises so that some vendors can use
 127057 historically limited versions of UNIX system utilities. They are the highest values that a strictly
 127058 conforming application can assume, given no other information.

127059 However, by using the *getconf* utility or the *sysconf()* function, the elegant application can be
 127060 tailored to more liberal values on some of the specific instances of specific implementations.

127061 There is no explicitly stated requirement that an implementation provide finite limits for any of
 127062 these numeric values; the implementation is free to provide essentially unbounded capabilities
 127063 (where it makes sense), stopping only at reasonable points such as {ULONG_MAX} (from the
 127064 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a
 127065 particular implementation need to be ready for possibly huge values; it may not be a good idea
 127066 to allocate blindly a buffer for an input line based on the value of {LINE_MAX}, for instance.
 127067 However, unlike the System Interfaces volume of POSIX.1-2017, there is no set of limits that
 127068 return a special indication meaning “unbounded”. The implementation should always return an
 127069 actual number, even if the number is very large.

127070 The statement:

```
127071 “It is not guaranteed that the application ...”
```

127072 is an indication that many of these limits are designed to ensure that implementors design their
 127073 utilities without arbitrary constraints related to unimaginative programming. There are certainly
 127074 conditions under which combinations of options can cause failures that would not render an
 127075 implementation non-conforming. For example, {EXPR_NEST_MAX} and {ARG_MAX} could
 127076 collide when expressions are large; combinations of {BC_SCALE_MAX} and {BC_DIM_MAX}
 127077 could exceed virtual memory.

127078 In the Shell and Utilities volume of POSIX.1-2017, the notion of a limit being guaranteed for the
 127079 process lifetime, as it is in the System Interfaces volume of POSIX.1-2017, is not as useful to a
 127080 shell script. The *getconf* utility is probably a process itself, so the guarantee would be without
 127081 value. Therefore, the Shell and Utilities volume of POSIX.1-2017 requires the guarantee to be for
 127082 the session lifetime. This will mean that many vendors will either return very conservative
 127083 values or possibly implement *getconf* as a built-in.

127084 It may seem confusing to have limits that apply only to a single utility grouped into one global
 127085 section. However, the alternative, which would be to disperse them out into their utility
 127086 description sections, would cause great difficulty when *sysconf()* and *getconf* were described.
 127087 Therefore, the standard developers chose the global approach.

127088 Each language binding could provide symbol names that are slightly different from those shown
 127089 here. For example, the C-Language Binding option adds a leading <underscore> to the symbols
 127090 as a prefix.

127091 The following comments describe selection criteria for the symbols and their values:

127092 {ARG_MAX}

127093 This is defined by the System Interfaces volume of POSIX.1-2017. Unfortunately, it is very

127094 difficult for a conforming application to deal with this value, as it does not know how much

127095 of its argument space is being consumed by the environment variables of the user.

127096 {BC_BASE_MAX}

127097 {BC_DIM_MAX}

127098 {BC_SCALE_MAX}

127099 These were originally one value, {BC_SCALE_MAX}, but it was unreasonable to link all

127100 three concepts into one limit.

127101 {CHILD_MAX}

127102 This is defined by the System Interfaces volume of POSIX.1-2017.

127103 {COLL_WEIGHTS_MAX}

127104 The weights assigned to **order** can be considered as “passes” through the collation

127105 algorithm.

127106 {EXPR_NEST_MAX}

127107 The value for expression nesting was borrowed from the ISO C standard.

127108 {LINE_MAX}

127109 This is a global limit that affects all utilities, unless otherwise noted. The {MAX_CANON}

127110 value from the System Interfaces volume of POSIX.1-2017 may further limit input lines from

127111 terminals. The {LINE_MAX} value was the subject of much debate and is a compromise

127112 between those who wished to have unlimited lines and those who understood that many

127113 historical utilities were written with fixed buffers. Frequently, utility writers selected the

127114 UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities were

127115 limited to 512 bytes for I/O lines, while others achieved 4 096 bytes or greater.

127116 It should be noted that {LINE_MAX} applies only to input line length; there is no

127117 requirement in POSIX.1-2017 that limits the length of output lines. Utilities such as *awk*, *sed*,

127118 and *paste* could theoretically construct lines longer than any of the input lines they received,

127119 depending on the options used or the instructions from the application. They are not

127120 required to truncate their output to {LINE_MAX}. It is the responsibility of the application

127121 to deal with this. If the output of one of those utilities is to be piped into another of the

127122 standard utilities, line length restrictions will have to be considered; the *fold* utility, among

127123 others, could be used to ensure that only reasonable line lengths reach utilities or

127124 applications.

127125 {LINK_MAX}

127126 This is defined by the System Interfaces volume of POSIX.1-2017.

127127 {MAX_CANON}

127128 {MAX_INPUT}

127129 {NAME_MAX}

127130 {NGROUPS_MAX}

127131 {OPEN_MAX}

127132 {PATH_MAX}

127133 {PIPE_BUF}

127134 These limits are defined by the System Interfaces volume of POSIX.1-2017. Note that the

127135 byte lengths described by some of these values continue to represent bytes, even if the

127136 applicable character set uses a multi-byte encoding.

127137 {RE_DUP_MAX}
 127138 The value selected is consistent with historical practice. Although the name implies that it
 127139 applies to all REs, only BREs use the interval notation $\{m,n\}$ addressed by this limit.

127140 {POSIX2_SYMLINKS}
 127141 The {POSIX2_SYMLINKS} variable indicates that the underlying operating system supports
 127142 the creation of symbolic links in specific directories. Many of the utilities defined in
 127143 POSIX.1-2017 that deal with symbolic links do not depend on this value. For example, a
 127144 utility that follows symbolic links (or does not, as the case may be) will only be affected by a
 127145 symbolic link if it encounters one. Presumably, a file system that does not support symbolic
 127146 links will not contain any. This variable does affect such utilities as *ln -s* and *pax* that
 127147 attempt to create symbolic links.

127148 There are different limits associated with command lines and input to utilities, depending on the
 127149 method of invocation. In the case of a C program *exec*-ing a utility, {ARG_MAX} is the
 127150 underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE_MAX}
 127151 limits the length of lines the shell is required to process, and {ARG_MAX} will still be a limit. If a
 127152 user is entering a command on a terminal to the shell, requesting that it invoke the utility,
 127153 {MAX_INPUT} may restrict the length of the line that can be given to the shell to a value below
 127154 {LINE_MAX}.

127155 When an option is supported, *getconf* returns a value of 1. For example, when C development is
 127156 supported:

```
127157 if [ "$(getconf POSIX2_C_DEV)" -eq 1 ]; then
127158     echo C supported
127159 fi
```

127160 The *sysconf()* function in the C-Language Binding option would return 1.

127161 The following comments describe selection criteria for the symbols and their values:

```
127162 POSIX2_C_BIND
127163 POSIX2_C_DEV
127164 POSIX2_FORT_DEV
127165 POSIX2_FORT_RUN
127166 POSIX2_SW_DEV
127167 POSIX2_UPE
```

127168 It is possible for some (usually privileged) operations to remove utilities that support these
 127169 options or otherwise to render these options unsupported. The header files, the *sysconf()*
 127170 function, or the *getconf* utility will not necessarily detect such actions, in which case they
 127171 should not be considered as rendering the implementation non-conforming. A test suite
 127172 should not attempt tests such as:

```
127173 rm /usr/bin/c99
127174 getconf POSIX2_C_DEV
```

```
127175 POSIX2_LOCALEDEF
```

127176 This symbol was introduced to allow implementations to restrict supported locales to only
 127177 those supplied by the implementation.

127178 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/2 is applied, deleting the entry for
 127179 {POSIX2_VERSION} since it is not a utility limit minimum value.

127180 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/3 is applied, changing the text in Utility
 127181 Limits from: ``utility (see *getconf*) through the *sysconf()* function defined in the System Interfaces
 127182 volume of POSIX.1-2017. The literal names shown in Table 1-3 apply only to the *getconf* utility;
 127183 the high-level language binding describes the exact form of each name to be used by the

127184 interfaces in that binding.” to: ``utility (see *getconf*).”.

127185 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0001 [666] is applied.

127186 **C.1.4 Grammar Conventions**

127187 There is no additional rationale provided for this section.

127188 **C.1.5 Utility Description Defaults**

127189 This section is arranged with headings in the same order as all the utility descriptions. It is a
127190 collection of related and unrelated information concerning:

- 127191 1. The default actions of utilities
- 127192 2. The meanings of notations used in POSIX.1-2017 that are specific to individual utility
127193 sections

127194 Although this material may seem out of place here, it is important that this information appear
127195 before any of the utilities to be described later.

127196 **NAME**

127197 There is no additional rationale provided for this section.

127198 **SYNOPSIS**

127199 There is no additional rationale provided for this section.

127200 **DESCRIPTION**

127201 There is no additional rationale provided for this section.

127202 **OPTIONS**

127203 Although it has not always been possible, the standard developers tried to avoid repeating
127204 information to reduce the risk that duplicate explanations could each be modified differently.

127205 The need to recognize `--` is required because conforming applications need to shield their
127206 operands from any arbitrary options that the implementation may provide as an extension. For
127207 example, if the standard utility *foo* is listed as taking no options, and the application needed to
127208 give it a pathname with a leading <hyphen-minus>, it could safely do it as:

```
127209 foo -- -myfile
```

127210 and avoid any problems with `-m` used as an extension.

127211 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0002 [584] is applied.

127212 **OPERANDS**

127213 The usage of `-` is never shown in the SYNOPSIS. Similarly, the usage of `--` is never shown.

127214 The requirement for processing operands in command-line order is to avoid a “WeirdNIX”
 127215 utility that might choose to sort the input files alphabetically, by size, or by directory order.
 127216 Although this might be acceptable for some utilities, in general the programmer has a right to
 127217 know exactly what order will be chosen.

127218 Some of the standard utilities take multiple *file* operands and act as if they were processing the
 127219 concatenation of those files. For example:

```
127220 asa file1 file2
127221 and:
```

```
127222 cat file1 file2 | asa
```

127223 have similar results when questions of file access, errors, and performance are ignored. Other
 127224 utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of
 127225 utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is
 127226 not. Although it might be possible to create a general assertion about the former case, the
 127227 following points must be addressed:

127228 Access times for the files might be different in the operand case *versus* the *cat* case.

127229 The utility may have error messages that are cognizant of the input filename, and this
 127230 added value should not be suppressed. (As an example, *awk* sets a variable with the
 127231 filename at each file boundary.)

127232 **STDIN**

127233 There is no additional rationale provided for this section.

127234 **INPUT FILES**

127235 A conforming application cannot assume the following three commands are equivalent:

```
127236 tail -n +2 file
127237 (sed -n 1q; cat) < file
127238 cat file | (sed -n 1q; cat)
```

127239 The second command is equivalent to the first only when the file is seekable. In the third
 127240 command, if the file offset in the open file description were not unspecified, *sed* would have to
 127241 be implemented so that it read from the pipe 1 byte at a time or it would have to employ some
 127242 method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1
 127243 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar
 127244 properties, so the restriction is described globally in this section.

127245 The definition of “text file” is strictly enforced for input to the standard utilities; very few of
 127246 them list exceptions to the undefined results called for here. (Of course, “undefined” here does
 127247 not mean that historical implementations necessarily have to change to start indicating error
 127248 conditions. Conforming applications cannot rely on implementations succeeding or failing when
 127249 non-text files are used.)

127250 The utilities that allow line continuation are generally those that accept input languages, rather
 127251 than pure data. It would be unusual for an input line of this type to exceed `{LINE_MAX}` bytes
 127252 and unreasonable to require that the implementation allow unlimited accumulation of multiple
 127253 lines, each of which could reach `{LINE_MAX}`. Thus, for a conforming application the total of all
 127254 the continued lines in a set cannot exceed `{LINE_MAX}`.

127255 The format description is intended to be sufficiently rigorous to allow other applications to
127256 generate these input files. However, since <blank> characters can legitimately be included in
127257 some of the fields described by the standard utilities, particularly in locales other than the POSIX
127258 locale, this intent is not always realized.

127259 **ENVIRONMENT VARIABLES**

127260 There is no additional rationale provided for this section.

127261 **ASYNCHRONOUS EVENTS**

127262 Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some
127263 additional processing (such as deleting temporary files), restore the default signal action (or
127264 action inherited from the parent process), and resignal itself.

127265 **STDOUT**

127266 The format description is intended to be sufficiently rigorous to allow post-processing of output
127267 by other programs, particularly by an *awk* or *lex* parser.

127268 **STDERR**

127269 This section does not describe error messages that refer to incorrect operation of the utility.
127270 Consider a utility that processes program source code as its input. This section is used to
127271 describe messages produced by a correctly operating utility that encounters an error in the
127272 program source code on which it is processing. However, a message indicating that the utility
127273 had insufficient memory in which to operate would not be described.

127274 Some utilities have traditionally produced warning messages without returning a non-zero exit
127275 status; these are specifically noted in their sections. Other utilities shall not write to standard
127276 error if they complete successfully, unless the implementation provides some sort of extension to
127277 increase the verbosity or debugging level.

127278 The format descriptions are intended to be sufficiently rigorous to allow post-processing of
127279 output by other programs.

127280 **OUTPUT FILES**

127281 The format description is intended to be sufficiently rigorous to allow post-processing of output
127282 by other programs, particularly by an *awk* or *lex* parser.

127283 Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging
127284 mode) that would bypass any attempted recovery actions.

127285 **EXTENDED DESCRIPTION**

127286 There is no additional rationale provided for this section.

127287 **EXIT STATUS**

127288 Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It
127289 describes requirements for returning exit values greater than 125.

127290 A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than
127291 1 as “an error occurred”. In this case, unspecified conditions may cause a 2 or 3, or other value,
127292 to be returned. A strictly conforming application should be written so that it tests for successful
127293 exit status values (zero in this case), rather than relying upon the single specific error value listed
127294 in POSIX.1-2017. In that way, it will have maximum portability, even on implementations with

127295 extensions.

127296 The standard developers are aware that the general non-enumeration of errors makes it difficult
127297 to write test suites that test the *incorrect* operation of utilities. There are some historical
127298 implementations that have expended effort to provide detailed status messages and a helpful
127299 environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant
127300 syntax errors; other implementations have not. Since there is no realistic way to mandate system
127301 behavior in cases of undefined application actions or system problems—in a manner acceptable
127302 to all cultures and environments—attention has been limited to the correct operation of utilities
127303 by the conforming application. Furthermore, the conforming application does not need detailed
127304 information concerning errors that it caused through incorrect usage or that it cannot correct.

127305 There is no description of defaults for this section because all of the standard utilities specify
127306 something (or explicitly state “Unspecified”) for exit status.

127307 **CONSEQUENCES OF ERRORS**

127308 Several actions are possible when a utility encounters an error condition, depending on the
127309 severity of the error and the state of the utility. Included in the possible actions of various
127310 utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and
127311 validity checking of the file system or directory.

127312 The text about recursive traversing is meant to ensure that utilities such as *find* process as many
127313 files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error
127314 and resume with the next command-line operand, but should attempt to keep going.

127315 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0001 [150] is applied.

127316 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0003 [913] is applied.

127317 **APPLICATION USAGE**

127318 This section provides additional caveats, issues, and recommendations to the developer.

127319 **EXAMPLES**

127320 This section provides sample usage.

127321 **RATIONALE**

127322 There is no additional rationale provided for this section.

127323 **FUTURE DIRECTIONS**

127324 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in
127325 the future, and often cautions the developer to architect the code to account for a change in this
127326 area. Note that a future directions statement should not be taken as a commitment to adopt a
127327 feature or interface in the future.

127328 **SEE ALSO**

127329 There is no additional rationale provided for this section.

127330 **CHANGE HISTORY**

127331 There is no additional rationale provided for this section.

127332 C.1.6 Considerations for Utilities in Support of Files of Arbitrary Size

127333 This section is intended to clarify the requirements for utilities in support of large files.

127334 The utilities listed in this section are utilities which are used to perform administrative tasks
 127335 such as to create, move, copy, remove, change the permissions, or measure the resources of a file.
 127336 They are useful both as end-user tools and as utilities invoked by applications during software
 127337 installation and operation.

127338 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file-capable versions of
 127339 *stat()*, *lstat()*, *ftw()*, and the **stat** structure.

127340 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, *sum*, and *touch* utilities probably require use of large file-capable
 127341 versions of *creat()*, *open()*, and *fopen()*.

127342 The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, *ls*, and *sum* utilities may require writing large integer values. For
 127343 example:

127344 The *cat* utility might have a `-n` option which counts <newline> characters.

127345 The *cksum* and *ls* utilities report file sizes.

127346 The *cmp* utility reports the line number at which the first difference occurs, and also has a
 127347 `-l` option which reports file offsets.

127348 The *dd*, *df*, *du*, *ls*, and *sum* utilities report block counts.

127349 The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit
 127350 values. For *dd*, the arguments include *skip=n*, *seek=n*, and *count=n*. For *find*, the arguments
 127351 include `-sizen`. For *test*, the arguments are those associated with algebraic comparisons.

127352 The *df* utility might need to access large file systems with *statvfs()*.

127353 The *ulimit* utility will need to use large file-capable versions of *getrlimit()* and *setrlimit()* and be
 127354 able to read and write large integer values.

127355 C.1.7 Built-In Utilities

127356 All of these utilities can be *exec*-ed. There is no requirement that these utilities are actually built
 127357 into the shell itself, but many shells need the capability to do so because XCU [Section 2.9.1.1](#) (on
 127358 page 2367) requires that they be found prior to the *PATH* search. The shell could satisfy its
 127359 requirements by keeping a list of the names and directly accessing the file-system versions
 127360 regardless of *PATH*. Providing all of the required functionality for those such as *cd* or *read* would
 127361 be more difficult.

127362 There were originally three justifications for allowing the omission of *exec*-able versions:

127363 1. It would require wasting space in the file system, at the expense of very small systems.
 127364 However, it has been pointed out that all 16 utilities in the table can be provided with 16
 127365 links to a single-line shell script:

127366 `$0 "$@"`

127367 2. It is not logical to require invocation of utilities such as *cd* because they have no value
 127368 outside the shell environment or cannot be useful in a child process. However, counter-
 127369 examples always seemed to be available for even the most unusual cases:

127370 `find . -type d -exec cd {} \; -exec foo {} \;`
 127371 (which invokes ``foo'' on accessible directories)

127372 `ps ... | sed ... | xargs kill`

127373 `find . -exec true \; -a ...`

127374 (where “true” is used for temporary debugging)

127375 3. It is confusing to have a utility such as *kill* that can easily be in the file system in the base
 127376 standard, but that requires built-in status for the User Portability Utilities option (for the
 127377 % job control job ID notation). It was decided that it was more appropriate to describe the
 127378 required functionality (rather than the implementation) to the system implementors and
 127379 let them decide how to satisfy it.

127380 On the other hand, it was realized that any distinction like this between utilities was not useful
 127381 to applications, and that the cost to correct it was small. These arguments were ultimately the
 127382 most effective.

127383 There were varying reasons for including utilities in the table of built-ins:

127384 *alias, fc, unalias*

127385 The functionality of these utilities is performed more simply within the shell itself and that
 127386 is the model most historical implementations have used.

127387 *bg, fg, jobs*

127388 All of the job control-related utilities are eligible for built-in status because that is the model
 127389 most historical implementations have used.

127390 *cd, getopts, newgrp, read, umask, wait*

127391 The functionality of these utilities is performed more simply within the context of the
 127392 current process. An example can be taken from the usage of the *cd* utility. The purpose of
 127393 the *cd* utility is to change the working directory for subsequent operations. The actions of *cd*
 127394 affect the process in which *cd* is executed and all subsequent child processes of that process.
 127395 Based on the POSIX standard process model, changes in the process environment of a child
 127396 process have no effect on the parent process. If the *cd* utility were executed from a child
 127397 process, the working directory change would be effective only in the child process. Child
 127398 processes initiated subsequent to the child process that executed the *cd* utility would not
 127399 have a changed working directory relative to the parent process.

127400 *command*

127401 This utility was placed in the table primarily to protect scripts that are concerned about
 127402 their *PATH* being manipulated. The “secure” shell script example in the *command* utility in
 127403 the Shell and Utilities volume of POSIX.1-2017 would not be possible if a *PATH* change
 127404 retrieved an alien version of *command*. (An alternative would have been to implement
 127405 *getconf* as a built-in, but the standard developers considered that it carried too many
 127406 changing configuration strings to require in the shell.)

127407 *kill* Since *kill* provides optional job control functionality using shell notation (%1, %2, and so on),
 127408 some implementations would find it extremely difficult to provide this outside the shell.

127409 *true, false*

127410 These are in the table as a courtesy to programmers who wish to use the “while true”
 127411 shell construct without protecting *true* from *PATH* searches. (It is acknowledged that
 127412 “while :” also works, but the idiom with *true* is historically pervasive.)

127413 All utilities, including those in the table, are accessible via the *system()* and *popen()* functions in
 127414 the System Interfaces volume of POSIX.1-2017. There are situations where the return
 127415 functionality of *system()* and *popen()* is not desirable. Applications that require the exit status of
 127416 the invoked utility will not be able to use *system()* or *popen()*, since the exit status returned is
 127417 that of the command language interpreter rather than that of the invoked utility. The alternative
 127418 for such applications is the use of the *exec* family.

127419 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0004 [705] is applied.

127420 C.2 Shell Command Language

127421 C.2.1 Shell Introduction

127422 The System V shell was selected as the starting point for the Shell and Utilities volume of
127423 POSIX.1-2017. The BSD C shell was excluded from consideration for the following reasons:

127424 Most historically portable shell scripts assume the Version 7 Bourne shell, from which the
127425 System V shell is derived.

127426 The majority of tutorial materials on shell programming assume the System V shell.

127427 The construct "#!" is reserved for implementations wishing to provide that extension. If it were
127428 not reserved, the Shell and Utilities volume of POSIX.1-2017 would disallow it by forcing it to be
127429 a comment. As it stands, a strictly conforming application must not use "#!" as the first two
127430 characters of the file.

127431 C.2.2 Quoting

127432 There is no additional rationale provided for this section.

127433 C.2.2.1 Escape Character (Backslash)

127434 There is no additional rationale provided for this section.

127435 C.2.2.2 Single-Quotes

127436 A <backslash> cannot be used to escape a single-quote in a single-quoted string. An embedded
127437 quote can be created by writing, for example: "'a'\''b'", which yields "a'b". (See XCU
127438 [Section 2.6.5](#) (on page 2359) for a better understanding of how portions of words are either split
127439 into fields or remain concatenated.) A single token can be made up of concatenated partial
127440 strings containing all three kinds of quoting or escaping, thus permitting any combination of
127441 characters.

127442 C.2.2.3 Double-Quotes

127443 The escaped <newline> used for line continuation is removed entirely from the input and is not
127444 replaced by any white space. Therefore, it cannot serve as a token separator.

127445 In double-quoting, if a <backslash> is immediately followed by a character that would be
127446 interpreted as having a special meaning, the <backslash> is deleted and the subsequent
127447 character is taken literally. If a <backslash> does not precede a character that would have a
127448 special meaning, it is left in place unmodified and the character immediately following it is also
127449 left unmodified. Thus, for example:

127450 "\$" -> \$

127451 "\a" -> \a

127452 It would be desirable to include the statement "The characters from an enclosed "\${" to the
127453 matching "}" shall not be affected by the double-quotes", similar to the one for "\$()".
127454 However, historical practice in the System V shell prevents this.

127455 The requirement that double-quotes be matched inside "\${...}" within double-quotes and the
 127456 rule for finding the matching '}' in XCU Section 2.6.2 (on page 2354) eliminate several subtle
 127457 inconsistencies in expansion for historical shells in rare cases; for example:

```
127458 "${foo-bar}"
```

127459 yields **bar** when **foo** is not defined, and is an invalid substitution when **foo** is defined, in many
 127460 historical shells. The differences in processing the "\${...}" form have led to inconsistencies
 127461 between historical systems. A consequence of this rule is that single-quotes cannot be used to
 127462 quote the '}' within "\${...}"; for example:

```
127463 unset bar  
127464 foo="${bar-'}'"
```

127465 is invalid because the "\${...}" substitution contains an unpaired unescaped single-quote. The
 127466 <backslash> can be used to escape the '}' in this example to achieve the desired result:

```
127467 unset bar  
127468 foo="${bar-\\}'"
```

127469 The differences in processing the "\${...}" form have led to inconsistencies between the
 127470 historical System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of
 127471 POSIX.1-2017 is an attempt to converge them without breaking too many applications. The only
 127472 alternative to this compromise between shells would be to make the behavior unspecified
 127473 whenever the literal characters single-quote, '{', '}', and '"' appear within "\${...}". To
 127474 write a portable script that uses these values, a user would have to assign variables; for example:

```
127475 squote=\' dquote=\" lbrace='{ ' rbrace='}'  
127476 ${foo-`$squote$rbrace$squote`}
```

127477 rather than:

```
127478 ${foo-"' }'"}
```

127479 Some implementations have allowed the end of the word to terminate the backquoted command
 127480 substitution, such as in:

```
127481 "`echo hello"
```

127482 This usage is undefined; the matching backquote is required by the Shell and Utilities volume of
 127483 POSIX.1-2017. The other undefined usage can be illustrated by the example:

```
127484 sh -c '` echo "foo`'
```

127485 The description of the recursive actions involving command substitution can be illustrated with
 127486 an example. Upon recognizing the introduction of command substitution, the shell parses input
 127487 (in a new context), gathering the source for the command substitution until an unbalanced ')' or
 127488 or '`' is located. For example, in the following:

```
127489 echo "$(date; echo "  
127490 one" )"
```

127491 the double-quote following the *echo* does not terminate the first double-quote; it is part of the
 127492 command substitution script. Similarly, in:

```
127493 echo "$(echo *)"
```

127494 the <asterisk> is not quoted since it is inside command substitution; however:

```
127495 echo "$(echo "*" )"
```

127496 is quoted (and represents the <asterisk> character itself).

127497 **C.2.3 Token Recognition**

127498 The "(" and ")" symbols are control operators in the KornShell, used for an alternative
 127499 syntax of an arithmetic expression command. A conforming application cannot use "(" as a
 127500 single token (with the exception of the "\$(" form for shell arithmetic).

127501 On some implementations, the symbol "(" is a control operator; its use produces unspecified
 127502 results. Applications that wish to have nested subshells, such as:

```
127503 ((echo Hello);(echo World))
```

127504 must separate the "(" characters into two tokens by including white space between them.
 127505 Some systems may treat these as invalid arithmetic expressions instead of subshells.

127506 Certain combinations of characters are invalid in portable scripts, as shown in the grammar.
 127507 Implementations may use these combinations (such as "|&") as valid control operators. Portable
 127508 scripts cannot rely on receiving errors in all cases where this volume of POSIX.1-2017 indicates
 127509 that a syntax is invalid.

127510 The (3) rule about combining characters to form operators is not meant to preclude systems from
 127511 extending the shell language when characters are combined in otherwise invalid ways.
 127512 Conforming applications cannot use invalid combinations, and test suites should not penalize
 127513 systems that take advantage of this fact. For example, the unquoted combination "|&" is not
 127514 valid in a POSIX script, but has a specific KornShell meaning.

127515 The (10) rule about '#' as the current character is the first in the sequence in which a new token
 127516 is being assembled. The '#' starts a comment only when it is at the beginning of a token. This
 127517 rule is also written to indicate that the search for the end-of-comment does not consider escaped
 127518 <newline> specially, so that a comment cannot be continued to the next line.

127519 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0005 [718], XCU/TC2-2008/0006
 127520 [647], XCU/TC2-2008/0007 [568], and XCU/TC2-2008/0008 [648] are applied.

127521 **C.2.3.1 Alias Substitution**

127522 The alias capability was added because it is widely used in historical implementations by
 127523 interactive users.

127524 The definition of "alias name" precludes an alias name containing a <slash> character. Since the
 127525 text applies to the command words of simple commands, reserved words (in their proper
 127526 places) cannot be confused with aliases.

127527 The placement of alias substitution in token recognition makes it clear that it precedes all of the
 127528 word expansion steps.

127529 An example concerning trailing <blank> characters and reserved words follows. If the user
 127530 types:

```
127531 $ alias foo="/bin/ls "  

  127532 $ alias while="/"
```

127533 The effect of executing:

```
127534 $ while true  

  127535 > do  

  127536 > echo "Hello, World"  

  127537 > done
```

127538 is a never-ending sequence of "Hello, World" strings to the screen. However, if the user
 127539 types:

127540 \$ foo while

127541 the result is an *ls* listing of */*. Since the alias substitution for **foo** ends in a <space>, the next word

127542 is checked for alias substitution. The next word, **while**, has also been aliased, so it is substituted

127543 as well. Since it is not in the proper position as a command word, it is not recognized as a

127544 reserved word.

127545 If the user types:

127546 \$ foo; while

127547 **while** retains its normal reserved-word properties.

127548 C.2.4 Reserved Words

127549 All reserved words are recognized syntactically as such in the contexts described. However, note

127550 that **in** is the only meaningful reserved word after a **case** or **for**; similarly, **in** is not meaningful as

127551 the first word of a simple command.

127552 Reserved words are recognized only when they are delimited (that is, meet the definition of XBD

127553 Section 3.446, on page 105), whereas operators are themselves delimiters. For instance, ' (' and

127554 ') ' are control operators, so that no <space> is needed in (*list*). However, '{ ' and ' } ' are

127555 reserved words in { *list*; }, so that in this case the leading <space> and <semicolon> are required.

127556 The list of unspecified reserved words is from the KornShell, so conforming applications cannot

127557 use them in places a reserved word would be recognized. This list contained **time** in early

127558 proposals, but it was removed when the *time* utility was selected for the Shell and Utilities

127559 volume of POSIX.1-2017.

127560 There was a strong argument for promoting braces to operators (instead of reserved words), so

127561 they would be syntactically equivalent to subshell operators. Concerns about compatibility

127562 outweighed the advantages of this approach. Nevertheless, conforming applications should

127563 consider quoting '{ ' and ' } ' when they represent themselves.

127564 The restriction on ending a name with a <colon> is to allow future implementations that support

127565 named labels for flow control; see the RATIONALE for the *break* built-in utility.

127566 It is possible that a future version of the Shell and Utilities volume of POSIX.1-2017 may require

127567 that '{ ' and ' } ' be treated individually as control operators, although the token "{ }" will

127568 probably be a special-case exemption from this because of the often-used *find*{ } construct.

127569 C.2.5 Parameters and Variables

127570 C.2.5.1 Positional Parameters

127571 There is no additional rationale provided for this section.

127572 C.2.5.2 Special Parameters

127573 Most historical implementations implement subshells by forking; thus, the special parameter

127574 '\$ ' does not necessarily represent the process ID of the shell process executing the commands

127575 since the subshell execution environment preserves the value of '\$ '.

127576 If a subshell were to execute a background command, the value of "\$!" for the parent would

127577 not change. For example:

```
127578 (
127579 date &
127580 echo $!
127581 )
127582 echo $!
```

127583 would echo two different values for "\$!".

127584 The "\$-" special parameter can be used to save and restore *set* options:

```
127585 Save=$(echo $- | sed 's/[ics]//g')
127586 ...
127587 set +aCefnuvx
127588 if [ -n "$Save" ]; then
127589     set -$Save
127590 fi
```

127591 The three options are removed using *sed* in the example because they may appear in the value of
127592 "\$-" (from the *sh* command line), but are not valid options to *set*.

127593 The descriptions of parameters '*' and '@' assume the reader is familiar with the field
127594 splitting discussion in XCU [Section 2.6.5](#) (on page 2359) and understands that portions of the
127595 word remain concatenated unless there is some reason to split them into separate fields.

127596 The following examples illustrate some of the ways in which '*' and '@' can be expanded:

```
127597 set "abc" "def ghi" "jkl"
127598 unset novar
127599 IFS=' ' # a space
127600 printf '%s\n' $*
127601 abc
127602 def
127603 ghi
127604 jkl
127605 printf '%s\n' "$*"
127606 abc def ghi jkl
127607 printf '%s\n' xx$*yy
127608 xxabc
127609 def
127610 ghi
127611 jklyy
127612 printf '%s\n' "xx$*yy"
127613 xxabc def ghi jklyy
127614 printf '%s\n' $@
127615 abc
127616 def
127617 ghi
127618 jkl
127619 printf '%s\n' "$@"
127620 abc
127621 def ghi
127622 jkl
127623 printf '%s\n' ${1+"$@"}
127624 abc
127625 def ghi
```

```

127626      jkl
127627      printf '%s\n' ${novar-"$@"}
127628      abc
127629      def ghi
127630      jkl
127631      printf '%s\n' xx$@yy
127632      xxabc
127633      def
127634      ghi
127635      jklyy
127636      printf '%s\n' "xx$@yy"
127637      xxabc
127638      def ghi
127639      jklyy
127640      printf '%s\n' $@$@
127641      abc
127642      def
127643      ghi
127644      jklabc
127645      def
127646      ghi
127647      jkl
127648      printf '%s\n' "$@$@"
127649      abc
127650      def ghi
127651      jklabc
127652      def ghi
127653      jkl
127654      IFS=':'
127655      printf '%s\n' "$*"
127656      abc: def ghi: jkl
127657      var=$*; printf '%s\n' "$var"
127658      abc: def ghi: jkl
127659      var="$*"; printf '%s\n' "$var"
127660      abc: def ghi: jkl
127661      unset var
127662      printf '%s\n' ${var-$*}
127663      abc
127664      def ghi
127665      jkl
127666      printf '%s\n' "${var-$*}"
127667      abc: def ghi: jkl
127668      printf '%s\n' ${var-"$*"}
127669      abc: def ghi: jkl
127670      printf '%s\n' ${var=$*}
127671      abc
127672      def ghi
127673      jkl
127674      printf 'var=%s\n' "$var"
127675      var=abc: def ghi: jkl
127676      unset var
127677      printf '%s\n' "${var=$*}"
127678      abc: def ghi: jkl

```

```

127679     printf 'var=%s\n' "$var"
127680     var=abc:def ghi:jkl

127681     IFS=' ' # null
127682     printf '%s\n' "$*"
127683     abcdef ghijkl
127684     var=$*; printf '%s\n' "$var"
127685     abcdef ghijkl
127686     var="$*"; printf '%s\n' "$var"
127687     abcdef ghijkl
127688     unset var
127689     printf '%s\n' ${var-$*}
127690     abcdef ghijkl
127691     printf '%s\n' "${var-$*}"
127692     abcdef ghijkl
127693     printf '%s\n' ${var-"$*"}
127694     abcdef ghijkl
127695     printf '%s\n' ${var=$*}
127696     abcdef ghijkl
127697     printf 'var=%s\n' "$var"
127698     var=abcdef ghijkl
127699     unset var
127700     printf '%s\n' "${var=$*}"
127701     abcdef ghijkl
127702     printf 'var=%s\n' "$var"
127703     var=abcdef ghijkl
127704     printf '%s\n' "$@"
127705     abc
127706     def ghi
127707     jkl

127708     unset IFS
127709     printf '%s\n' "$*"
127710     abc def ghi jkl
127711     var=$*; printf '%s\n' "$var"
127712     abc def ghi jkl
127713     var="$*"; printf '%s\n' "$var"
127714     abc def ghi jkl
127715     unset var
127716     printf '%s\n' ${var-$*}
127717     abc
127718     def
127719     ghi
127720     jkl
127721     printf '%s\n' "${var-$*}"
127722     abc def ghi jkl
127723     printf '%s\n' ${var-"$*"}
127724     abc def ghi jkl
127725     printf '%s\n' ${var=$*}
127726     abc
127727     def
127728     ghi
127729     jkl
127730     printf 'var=%s\n' "$var"

```

```

127731     var=abc def ghi jkl
127732     unset var
127733     printf '%s\n' "${var=*$}"
127734     abc def ghi jkl
127735     printf 'var=%s\n' "$var"
127736     var=abc def ghi jkl
127737     printf '%s\n' "$@"
127738     abc
127739     def ghi
127740     jkl

127741     set one "" three
127742     printf "[%s]\n" $*
127743     [one]
127744     [] (this line of output is optional)
127745     [three]
127746     printf "[%s]\n" $@
127747     [one]
127748     [] (this line of output is optional)
127749     [three]

127750     set --
127751     printf "[%s]\n" foo "$*"
127752     [foo]
127753     []
127754     printf "[%s]\n" foo "$novar*$(echo)"
127755     [foo]
127756     []
127757     printf "[%s]\n" foo $@
127758     [foo]
127759     printf "[%s]\n" foo "$@"
127760     [foo]
127761     printf "[%s]\n" foo '$@'
127762     [foo]
127763     []
127764     printf "[%s]\n" foo '"$@"
127765     [foo]
127766     []
127767     printf "[%s]\n" foo "$novar$@$$(echo)"
127768     [foo]
127769     [] (this line of output is optional)
127770     printf "[%s]\n" foo '"$novar$@$$(echo)"
127771     [foo]
127772     []

```

127773 In all of the following commands the results of the expansion of '@' (if performed) are
127774 unspecified:

```

127775     var=$@
127776     var="$@"
127777     printf '%s\n' ${var=$@}
127778     printf '%s\n' "${var=$@}"
127779     printf '%s\n' ${var="$@"}
127780     printf '%s\n' ${var?@$}
127781     printf '%s\n' "${var?@$}"

```

```

127782     printf '%s\n' "${var?"$@"}
127783     printf '%s\n' "${#@}
127784     printf '%s\n' "${#}"
127785     printf '%s\n' "${@%foo}
127786     printf '%s\n' "${@%foo}"
127787     printf '%s\n' "${#@%foo}
127788     printf '%s\n' "${#@%foo}"
127789     printf '%s\n' "${var%$@}
127790     printf '%s\n' "${var%$@}"
127791     printf '%s\n' "${var%"$@"}
127792     printf '%s\n' "${var%%$@}
127793     printf '%s\n' "${var%%$@}"
127794     printf '%s\n' "${var%%%"$@"}
127795     printf '%s\n' "${var#$@}
127796     printf '%s\n' "${var#$@}"
127797     printf '%s\n' "${var#%"$@"}
127798     printf '%s\n' "${var##$@}
127799     printf '%s\n' "${var##%"$@"}
127800     printf '%s\n' "${var##%"$@"}

```

127801 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0009 [888] is applied.

127802 C.2.5.3 Shell Variables

127803 See the discussion of *IFS* in [Section C.2.6.5](#) (on page 3734) and the RATIONALE for the *sh* utility.

127804 The prohibition on *LC_CTYPE* changes affecting lexical processing protects the shell
 127805 implementor (and the shell programmer) from the ill effects of changing the definition of
 127806 <blank> or the set of alphabetic characters in the current environment. It would probably not be
 127807 feasible to write a compiled version of a shell script without this rule. The rule applies only to
 127808 the current invocation of the shell and its subshells—invoking a shell script or performing *exec*
 127809 *sh* would subject the new shell to the changes in *LC_CTYPE*.

127810 Other common environment variables used by historical shells are not specified by the Shell and
 127811 Utilities volume of POSIX.1-2017, but they should be reserved for the historical uses.

127812 Tilde expansion for components of *PATH* in an assignment such as:

```
127813 PATH=~hlj/bin:~dwc/bin:$PATH
```

127814 is a feature of some historical shells and is allowed by the wording of XCU [Section 2.6.1](#) (on page
 127815 2354). Note that the <tilde> characters are expanded during the assignment to *PATH*, not when
 127816 *PATH* is accessed during command search.

127817 The following entries represent additional information about variables included in the Shell and
 127818 Utilities volume of POSIX.1-2017, or rationale for common variables in use by shells that have
 127819 been excluded:

127820 — (Underscore.) While <underscore> is historical practice, its overloaded usage
 127821 in the KornShell is confusing, and it has been omitted from the Shell and
 127822 Utilities volume of POSIX.1-2017.

127823 *ENV* This variable can be used to set aliases and other items local to the invocation
 127824 of a shell. The file referred to by *ENV* differs from **\$HOME/.profile** in that
 127825 **.profile** is typically executed at session start-up, whereas the *ENV* file is
 127826 executed at the beginning of each shell invocation. The *ENV* value is
 127827 interpreted in a manner similar to a dot script, in that the commands are

127828		executed in the current environment and the file needs to be readable, but not executable. However, unlike dot scripts, no <i>PATH</i> searching is performed. This is used as a guard against Trojan Horse security breaches.
127829		
127830		
127831	<i>ERRNO</i>	This variable was omitted from the Shell and Utilities volume of POSIX.1-2017 because the values of error numbers are not defined in POSIX.1-2017 in a portable manner.
127832		
127833		
127834	<i>FCEDIT</i>	Since this variable affects only the <i>fc</i> utility, it has been omitted from this more global place. The value of <i>FCEDIT</i> does not affect the command-line editing mode in the shell; see the description of <i>set -o vi</i> in the <i>set</i> built-in utility.
127835		
127836		
127837	<i>PS1</i>	This variable is used for interactive prompts. Historically, the “superuser” has had a prompt of '#'. Since privileges are not required to be monolithic, it is difficult to define which privileges should cause the alternate prompt. However, a sufficiently powerful user should be reminded of that power by having an alternate prompt.
127838		
127839		
127840		
127841		
127842	<i>PS3</i>	This variable is used by the KornShell for the <i>select</i> command. Since the POSIX shell does not include <i>select</i> , <i>PS3</i> was omitted.
127843		
127844	<i>PS4</i>	This variable is used for shell debugging. For example, the following script:
127845		<pre>PS4=' [\${LINENO}]+ '</pre>
127846		<pre>set -x</pre>
127847		<pre>echo Hello</pre>
127848		writes the following to standard error:
127849		<pre>[3]+ echo Hello</pre>
127850	<i>RANDOM</i>	This pseudo-random number generator was not seen as being useful to interactive users.
127851		
127852	<i>SECONDS</i>	Although this variable is sometimes used with <i>PS1</i> to allow the display of the current time in the prompt of the user, it is not one that would be manipulated frequently enough by an interactive user to include in the Shell and Utilities volume of POSIX.1-2017.
127853		
127854		
127855		
127856		POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0002 [152] is applied.
127857		POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0010 [888], XCU/TC2-2008/0011 [884], and XCU/TC2-2008/0012 [494] are applied.
127858		

127859 C.2.6 Word Expansions

127860	Step (2) refers to the “portions of fields generated by step (1)”. For example, if the word being expanded were "\$x+\$y" and <i>IFS</i> =+, the word would be split only if "\$x" or "\$y" contained
127861	'+'; the '+' in the original word was not generated by step (1).
127862	
127863	<i>IFS</i> is used for performing field splitting on the results of parameter and command substitution; it is not used for splitting all fields. Earlier versions of the shell used it for splitting all fields
127864	during field splitting, but this has severe problems because the shell can no longer parse its own
127865	script. There are also important security implications caused by this behavior. All useful
127866	applications of <i>IFS</i> use it for parsing input of the <i>read</i> utility and for splitting the results of
127867	parameter and command substitution.
127868	
127869	The rule concerning expansion to a single field requires that if foo=abc and bar=def , that:
127870	<pre>"\$foo"\$bar"</pre>

127871 expands to the single field:

127872 abcdef

127873 The rule concerning empty fields can be illustrated by:

```
127874 $ unset foo
127875 $ set $foo bar ' ' xyz "$foo" abc
127876 $ for i
127877 > do
127878 >     echo "-$i-"
127879 > done
127880 -bar-
127881 --
127882 -xyz-
127883 --
127884 -abc-
```

127885 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion
127886 are all processed simultaneously as they are scanned. For example, the following is valid
127887 arithmetic:

```
127888 x=1
127889 echo $(( $(echo 3)+$x ))
```

127890 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in
127891 known historical implementations; if it were, and if a referenced home directory contained a '\$'
127892 character, expansions would result within the directory name.

127893 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0003 [49,430] is applied.

127894 C.2.6.1 Tilde Expansion

127895 Tilde expansion generally occurs only at the beginning of words, but an exception based on
127896 historical practice has been included:

```
127897 PATH=/posix/bin:~djk/bin
```

127898 This is eligible for tilde expansion because <tilde> follows a <colon> and none of the relevant
127899 characters is quoted. Consideration was given to prohibiting this behavior because any of the
127900 following are reasonable substitutes:

```
127901 PATH=$(printf %s ~karels/bin : ~bostic/bin)
127902 for Dir in ~maat/bin ~srb/bin ...
127903 do
127904     PATH=${PATH:+$PATH:}$Dir
127905 done
```

127906 In the first command, explicit <colon> characters are used for each directory. In all cases, the
127907 shell performs tilde expansion on each directory because all are separate words to the shell.

127908 Note that expressions in operands such as:

```
127909 make -k mumble LIBDIR=~chet/lib
```

127910 do not qualify as shell variable assignments, and tilde expansion is not performed (unless the
127911 command does so itself, which *make* does not).

127912 Because of the requirement that the word is not quoted, the following are not equivalent; only
127913 the last causes tilde expansion:

127914 \`~hlj/` \`~h\lj/` \`~"hlj"/` \`~hlj\` \`~hlj/`

127915 In an early proposal, tilde expansion occurred following any unquoted `<equals-sign>` or
127916 `<colon>`, but this was removed because of its complexity and to avoid breaking commands such
127917 as:

127918 `rcp hostname:~marc/.profile .`

127919 System administrators on systems where `//` has an implementation-defined meaning which is
127920 different to `/`, should not create users with a home directory of `/` or `//`, since this may lead to
127921 unexpected filename resolution on those systems.

127922 A suggestion was made that the special sequence `"$~"` should be allowed to force tilde
127923 expansion anywhere. Since this is not historical practice, it has been left for future
127924 implementations to evaluate. (The description in XCU [Section 2.2](#) (on page 2346) requires that a
127925 `<dollar-sign>` be quoted to represent itself, so the `"$~"` combination is already unspecified.)

127926 The results of giving `<tilde>` with an unknown login name are undefined because the KornShell
127927 `"~+"` and `"~-"` constructs make use of this condition, but in general it is an error to give an
127928 incorrect login name with `<tilde>`. The results of having `HOME` unset are unspecified because
127929 some historical shells treat this as an error.

127930 Historically, the Korn shell performed field splitting and pathname expansion on the results of
127931 tilde expansion, and earlier versions of this standard reflected this. However, tilde expansion
127932 results in a pathname, and performing field splitting and pathname expansion on something
127933 that is already a pathname is at best redundant and at worst will change the value from the
127934 correct pathname to one or more incorrect ones. Later versions of the Korn shell do not perform
127935 these expansions and POSIX.1-2017 has been updated to match. Note that although pathname
127936 expansion is not performed on the results of tilde expansion, this does not prevent other parts of
127937 the same word from being expanded. For example, `~/a*` expands to all files in `$HOME`
127938 beginning with `'a'`.

127939 C.2.6.2 *Parameter Expansion*

127940 The rule for finding the closing `'}'` in `"${...}"` is the one used in the KornShell and is
127941 upwardly-compatible with the Bourne shell, which does not determine the closing `'}'` until the
127942 word is expanded. The advantage of this is that incomplete expansions, such as:

127943 ``${foo`

127944 can be determined during tokenization, rather than during expansion.

127945 For rationale regarding expansion of `"${...}"` within double-quotes, see [Section C.2.2.3](#) (on
127946 page 3718).

127947 The string length and substring capabilities were included because of the demonstrated need for
127948 them, based on their usage in other shells, such as C shell and KornShell.

127949 Historical versions of the KornShell have not performed tilde expansion on the word part of
127950 parameter expansion; however, it is more consistent to do so.

127951 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0004 [458], XCU/TC1-2008/0005
127952 [458], XCU/TC1-2008/0006 [457], XCU/TC1-2008/0007 [457], XCU/TC1-2008/0008 [417],
127953 XCU/TC1-2008/0009 [457], XCU/TC1-2008/0010 [457], XCU/TC1-2008/0011 [457],
127954 XCU/TC1-2008/0012 [457], XCU/TC1-2008/0013 [457], XCU/TC1-2008/0014 [457],
127955 XCU/TC1-2008/0015 [457], XCU/TC1-2008/0016 [457], XCU/TC1-2008/0017 [457], and
127956 XCU/TC1-2008/0018 [458] are applied.

127957 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0013 [888] and XCU/TC2-2008/0014

127958 [867] are applied.

127959 C.2.6.3 *Command Substitution*

127960 The "\$ ()" form of command substitution solves a problem of inconsistent behavior when using
127961 backquotes. For example:

127962	Command	Output
127963	echo '\\$x'	\\$x
127964	echo `echo '\\$x'`	\$x
127965	echo \$(echo '\\$x')	\\$x

127966 Additionally, the backquoted syntax has historical restrictions on the contents of the embedded
127967 command. While the newer "\$ ()" form can process any kind of valid embedded script, the
127968 backquoted form cannot handle some valid scripts that include backquotes. For example, these
127969 otherwise valid embedded scripts do not work in the left column, but do work on the right:

127970	echo `	echo \$(
127971	cat <<\eof	cat <<\eof
127972	a here-doc with `	a here-doc with)
127973	eof	eof
127974	`)
127975	echo `	echo \$(
127976	echo abc # a comment with `	echo abc # a comment with)
127977	`)
127978	echo `	echo \$(
127979	echo ` `	echo ` `)
127980	`)

127981 Because of these inconsistent behaviors, the backquoted variety of command substitution is not
127982 recommended for new applications that nest command substitutions or attempt to embed
127983 complex scripts.

127984 The KornShell feature:

127985 If *command* is of the form `<word`, *word* is expanded to generate a pathname, and the value
127986 of the command substitution is the contents of this file with any trailing `<newline>`
127987 characters deleted.

127988 was omitted from the Shell and Utilities volume of POSIX.1-2017 because `$(cat word)` is an
127989 appropriate substitute. However, to prevent breaking numerous scripts relying on this feature, it
127990 is unspecified to have a script within "\$ ()" that has only redirections.

127991 Arithmetic expansions have precedence over command substitutions. That is, if the shell can
127992 parse an expansion beginning with "\$ ((" as an arithmetic expansion then it will do so. It will
127993 only parse the expansion as a command substitution (that starts with a subshell) if it determines
127994 that it cannot parse the expansion as an arithmetic expansion. If the syntax is valid for neither
127995 type of expansion, then it is unspecified what kind of syntax error the shell reports.

127996 How well the shell performs this determination is a quality of implementation issue. Current
127997 shell implementations use heuristics. In particular, the shell need not evaluate nested expansions
127998 when determining whether it can parse an expansion beginning with "\$ ((" as an arithmetic
127999 expansion. For example:

128000 `$((a $op b))`

128001 is always an arithmetic expansion if "\$op" expands to, say, '+', but if "\$op" expands to '('
 128002 then the shell might still parse the expansion as an arithmetic expansion (resulting in a syntax
 128003 error due to unbalanced parentheses) or it might perform a command substitution.

128004 This standard requires that conforming applications always separate the "\$(" and '(' with
 128005 white space when a command substitution starts with a subshell. This is because
 128006 implementations may support extensions in arithmetic expressions which could result in the
 128007 shell parsing the input as an arithmetic expansion even though a minimally conforming shell
 128008 would not. For example, many shells support arrays with the array index (which can be an
 128009 expression) in square brackets. Therefore, the presence of "myfile[0-9]" within an expansion
 128010 beginning "\$(" is no guarantee that it will be parsed as a command substitution.

128011 The ambiguity is not restricted to the simple case of a single subshell. More complicated
 128012 ambiguous cases are possible (even with just the standard shell syntax), such as:

```
128013 $(( cat <<EOH
128014 + ( (
128015 EOH
128016 ) && ( cat <<EOH
128017 ) ) + 1 +
128018 EOH
128019 ) )
```

128020 This can be parsed as an arithmetic expansion, with *cat* and *EOH* as the names of shell variables.
 128021 Ambiguous cases also exist where the end of the expansion is at a different location for the
 128022 arithmetic expansion and the command substitution:

```
128023 $((cat <<EOF
128024 +(((
128025 EOF
128026 ) && (
128027 cat <<EOF
128028 +
128029 EOF
128030 ) )
```

128031 This is an incomplete arithmetic expansion, but would have been a (complete) command
 128032 substitution if it could not have been parsed as an arithmetic expansion. If this expansion occurs
 128033 at the end of input then the shell reports a syntax error; it does not parse it as a command
 128034 substitution.

128035 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/4 is applied, changing the text from: "If a
 128036 command substitution occurs inside double-quotes, it shall not be performed on the results of
 128037 the substitution." to: "If a command substitution occurs inside double-quotes, field splitting and
 128038 pathname expansion shall not be performed on the results of the substitution.". The
 128039 replacement text taken from the ISO POSIX-2:1993 standard is clearer about the items that are
 128040 not performed.

128041 SD5-XCU-ERN-84 is applied, clarifying how the search for the matching backquote is satisfied.

128042 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0019 [217] is applied.

128043 C.2.6.4 Arithmetic Expansion

128044 The standard developers agreed that there was a strong desire for some kind of arithmetic
 128045 evaluator to provide functionality similar to *expr*, that relating it to '\$' makes it work well with
 128046 the standard shell language and provides access to arithmetic evaluation in places where
 128047 accessing a utility would be inconvenient.

128048 The syntax and semantics for arithmetic were revised for the ISO/IEC 9945-2:1993 standard.
 128049 The language represents a simple subset of the previous arithmetic language (which was
 128050 derived from the KornShell "(())" construct). The syntax was changed from that of a
 128051 command denoted by *((expression))* to an expansion denoted by *\$(expression)*. The new form is
 128052 a dollar expansion ('\$') that evaluates the expression and substitutes the resulting value.
 128053 Objections to the previous style of arithmetic included that it was too complicated, did not fit in
 128054 well with the use of variables in the shell, and its syntax conflicted with subshells. The
 128055 justification for the new syntax is that the shell is traditionally a macro language, and if a new
 128056 feature is to be added, it should be accomplished by extending the capabilities presented by the
 128057 current model of the shell, rather than by inventing a new one outside the model; adding a new
 128058 dollar expansion was perceived to be the most intuitive and least destructive way to add such a
 128059 new capability.

128060 The standard requires assignment operators to be supported (as listed in XCU Section 1.1.2, on
 128061 page 2331), and since arithmetic expansions are not specified to be evaluated in a subshell
 128062 environment, changes to variables there have to be in effect after the arithmetic expansion, just
 128063 as in the parameter expansion "\${x=value}".

128064 Note, however, that "\${(x=5)}" need not be equivalent to "\${(\$x=5)}". If the value of
 128065 the environment variable *x* is the string "*y*", the expansion of "\${(x=5)}" would set *x* to 5
 128066 and output 5, but "\${(\$x=5)}" would output 0 if the value of the environment variable *y* is
 128067 not 5 and would output 1 if the environment variable *y* is 5. Similarly, if the value of the
 128068 environment variable is 4, the expansion of "\${(x=5)}" would still set *x* to 5 and output 5,
 128069 but "\${(\$x=5)}" (which would be equivalent to "\${(4=5)}") would yield a syntax
 128070 error.

128071 In early proposals, a form *\$(expression)* was used. It was functionally equivalent to the "\${()}"
 128072 of the current text, but objections were lodged that the 1988 KornShell had already implemented
 128073 "\${()}" and there was no compelling reason to invent yet another syntax. Furthermore, the
 128074 "\${[]}" syntax had a minor incompatibility involving the patterns in **case** statements.

128075 The portion of the ISO C standard arithmetic operations selected corresponds to the operations
 128076 historically supported in the KornShell. In addition to the exceptions listed in XCU Section 2.6.4
 128077 (on page 2358), the use of the following are explicitly outside the scope of the rules defined in
 128078 XCU Section 1.1.2.1 (on page 2331):

128079 The prefix operator '&' and the "[]", "->", and '.' operators.

128080 Casts

128081 It was concluded that the *test* command (I) was sufficient for the majority of relational arithmetic
 128082 tests, and that tests involving complicated relational expressions within the shell are rare, yet
 128083 could still be accommodated by testing the value of "\${()}" itself. For example:

```
128084 # a complicated relational expression
128085 while [ $(( (($x + $y)/($a * $b)) < ($foo*$bar) )) -ne 0 ]
```

128086 or better yet, the rare script that has many complex relational expressions could define a
 128087 function like this:

```
128088 val() {
128089     return $((!$1))
```

128090 }

128091 and complicated tests would be less intimidating:

```
128092            while val $(( (($x + $y)/($a * $b)) < ($foo*$bbar) ))
128093            do
128094                # some calculations
128095            done
```

128096 A suggestion that was not adopted was to modify *true* and *false* to take an optional argument, and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the argument was non-zero:

```
128099            while true $((($x > 5 && $y <= 25))
```

128100 There is a minor portability concern with the new syntax. The example "`$((2+2))`" could have been intended to mean a command substitution of a utility named "2+2" in a subshell. The standard developers considered this to be obscure and isolated to some KornShell scripts (because "`$()`" command substitution existed previously only in the KornShell). The text on command substitution requires that the "`$(`" and "`'`" be separate tokens if this usage is needed.

128106 An example such as:

```
128107            echo $((echo hi);(echo there))
```

128108 should not be misinterpreted by the shell as arithmetic because attempts to balance the parentheses pairs would indicate that they are subshells. However, as indicated by XBD [Section 3.113](#) (on page 51), a conforming application must separate two adjacent parentheses with white space to indicate nested subshells.

128112 The standard is intentionally silent about how a variable's numeric value in an expression is determined from its normal "sequence of bytes" value. It could be done as a text substitution, as a conversion like that performed by *strtol()*, or even recursive evaluation. Therefore, the only cases for which the standard is clear are those for which both conversions produce the same result. The cases where they give the same result are those where the sequence of bytes form a valid integer constant. Therefore, if a variable does not contain a valid integer constant, the behavior is unspecified.

128119 For the commands:

```
128120            x=010; echo $((x += 1))
```

128121 the output must be 9.

128122 For the commands:

```
128123            x=' 1'; echo $((x += 1))
```

128124 the results are unspecified.

128125 For the commands:

```
128126            x=1+1; echo $((x += 1))
```

128127 the results are unspecified.

128128 Although the ISO/IEC 9899:1999 standard now requires support for **long long** and allows extended integer types with higher ranks, POSIX.1-2017 only requires arithmetic expansions to support **signed long** integer arithmetic. Implementations are encouraged to support signed integer values at least as large as the size of the largest file allowed on the implementation.

128132 Implementations are also allowed to perform floating-point evaluations as long as an

- 128133 application won't see different results for expressions that would not overflow **signed long**
 128134 integer expression evaluation. (This includes appropriate truncation of results to integer values.)
- 128135 Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement
 128136 that the integer constant suffixes `l` and `L` had to be recognized. The ISO POSIX-2: 1993 standard
 128137 did not require the `u`, `ul`, `uL`, `U`, `Ul`, `UL`, `lu`, `lU`, `Lu`, and `LU` suffixes since only signed integer
 128138 arithmetic was required. Since all arithmetic expressions were treated as handling **signed long**
 128139 integer types anyway, the `l` and `L` suffixes were redundant. No known scripts used them and
 128140 some historic shells did not support them. When the ISO/IEC 9899: 1999 standard was used as
 128141 the basis for the description of arithmetic processing, the `ll` and `LL` suffixes and combinations
 128142 were also not required. Implementations are still free to accept any or all of these suffixes, but
 128143 are not required to do so.
- 128144 There was also some confusion as to whether the shell was required to recognize character
 128145 constants. Syntactically, character constants were required to be recognized, but the
 128146 requirements for the handling of `<backslash>` and single-quote characters (needed to specify
 128147 character constants) within an arithmetic expansion were ambiguous. Furthermore, no known
 128148 shells supported them. Changes made in response to IEEE PASC Interpretation 1003.2 #208
 128149 removed the requirement to support them (if they were indeed required before). POSIX.1-2017
 128150 clearly does not require support for character constants.
- 128151 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/3 is applied, clarifying arithmetic
 128152 expressions.
- 128153 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0020 [50] is applied.
- 128154 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0015 [584] is applied.
- 128155 **C.2.6.5** *Field Splitting*
- 128156 The operation of field splitting using *IFS*, as described in early proposals, was based on the way
 128157 the KornShell splits words, but it is incompatible with other common versions of the shell.
 128158 However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset
 128159 or is `<space><tab><newline>`, the operation is equivalent to the way the System V shell splits
 128160 words. Using characters outside the `<space><tab><newline>` set yields the KornShell behavior,
 128161 where each of the non-`<space><tab><newline>`s is significant. This behavior, which affords the
 128162 most flexibility, was taken from the way the original *awk* handled field splitting.
- 128163 Rule (3) can be summarized as a pseudo-ERE:
- 128164 $(s^*nS^* | s^+)$
- 128165 where *s* is an *IFS* white-space character and *n* is a character in the *IFS* that is not white space.
 128166 Any string matching that ERE delimits a field, except that the *s+* form does not delimit fields at
 128167 the beginning or the end of a line. For example, if *IFS* is `<space>/<comma>/<tab>`, the string:
- 128168 `<space><space>red<space><space>, <space>white<space>blue`
- 128169 yields the three colors as the delimited fields.
- 128170 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0016 [832] is applied.

128171 C.2.6.6 *Pathname Expansion*

128172 There is no additional rationale provided for this section.

128173 C.2.6.7 *Quote Removal*

128174 There is no additional rationale provided for this section.

128175 **C.2.7 Redirection**

128176 In the System Interfaces volume of POSIX.1-2017, file descriptors are integers in the range
128177 0–(OPEN_MAX–1). The file descriptors discussed in XCU [Section 2.7](#) (on page 2360) are that
128178 same set of small integers.

128179 Having multi-digit file descriptor numbers for I/O redirection can cause some obscure
128180 compatibility problems. Specifically, scripts that depend on an example command:

128181 `echo 22>/dev/null`

128182 echoing "2" to standard error or "22" to standard output are no longer portable. However, the
128183 file descriptor number must still be delimited from the preceding text. For example:

128184 `cat file2>foo`

128185 writes the contents of **file2**, not the contents of **file**.

128186 The ">|" format of output redirection was adopted from the KornShell. Along with the
128187 *noclobber* option, set `-C`, it provides a safety feature to prevent inadvertent overwriting of
128188 existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The
128189 restriction on regular files is historical practice.

128190 The System V shell and the KornShell have differed historically on pathname expansion of *word*;
128191 the former never performed it, the latter only when the result was a single field (file). As a
128192 compromise, it was decided that the KornShell functionality was useful, but only as a shorthand
128193 device for interactive users. No reasonable shell script would be written with a command such
128194 as:

128195 `cat foo > a*`

128196 Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with
128197 which they are most comfortable.

128198 The construct "`2>&1`" is often used to redirect standard error to the same file as standard
128199 output. Since the redirections take place beginning to end, the order of redirections is significant.
128200 For example:

128201 `ls > foo 2>&1`

128202 directs both standard output and standard error to file **foo**. However:

128203 `ls 2>&1 > foo`

128204 only directs standard output to file **foo** because standard error was duplicated as standard
128205 output before standard output was directed to file **foo**.

128206 Applications should not use the `[n]<&-` or `[n]>&-` operators to execute a utility or application
128207 with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, as
128208 this might cause the executed program (or shell built-in) to misbehave. In order not to pass on
128209 these file descriptors to an executed utility or application, applications should not just close
128210 them but should reopen them on, for example, `/dev/null`. Some implementations may reopen

128211 them automatically, but applications should not rely on this being done.

128212 The "<>" operator could be useful in writing an application that worked with several terminals,
 128213 and occasionally wanted to start up a shell. That shell would in turn be unable to run
 128214 applications that run from an ordinary controlling terminal unless it could make use of "<>"
 128215 redirection. The specific example is a historical version of the pager *more*, which reads from
 128216 standard error to get its commands, so standard input and standard output are both available
 128217 for their usual usage. There is no way of saying the following in the shell without "<>":

```
128218 cat food | more - >/dev/tty03 2<>/dev/tty03
```

128219 Another example of "<>" is one that opens **/dev/tty** on file descriptor 3 for reading and writing:

```
128220 exec 3<> /dev/tty
```

128221 An example of creating a lock file for a critical code region:

```
128222 set -C
128223 until      2> /dev/null > lockfile
128224 do        sleep 30
128225 done
128226 set +C
128227 perform critical function
128228 rm lockfile
```

128229 Since **/dev/null** is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

128230 Tilde expansion is not performed on a here-document because the data is treated as if it were
 128231 enclosed in double-quotes.

128232 C.2.7.1 *Redirecting Input*

128233 There is no additional rationale provided for this section.

128234 C.2.7.2 *Redirecting Output*

128235 There is no additional rationale provided for this section.

128236 C.2.7.3 *Appending Redirected Output*

128237 Note that when a file is opened (even with the `O_APPEND` flag set), the initial file offset for that
 128238 file is set to the beginning of the file. Some historic shells set the file offset to the current end-of-
 128239 file when **append** mode shell redirection was used, but this is not allowed by POSIX.1-2017.

128240 C.2.7.4 *Here-Document*

128241 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0017 [890], XCU/TC2-2008/0018
 128242 [583], and XCU/TC2-2008/0019 [580] are applied.

128243 C.2.7.5 *Duplicating an Input File Descriptor*

128244 There is no additional rationale provided for this section.

128245 C.2.7.6 *Duplicating an Output File Descriptor*

128246 There is no additional rationale provided for this section.

128247 C.2.7.7 *Open File Descriptors for Reading and Writing*

128248 There is no additional rationale provided for this section.

128249 C.2.8 Exit Status and Errors

128250 There is no additional rationale provided for this section.

128251 C.2.8.1 *Consequences of Shell Errors*

128252 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0020 [882] and XCU/TC2-2008/0021
128253 [717,882] are applied.

128254 C.2.8.2 *Exit Status for Commands*

128255 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command
128256 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues
128257 with the next command. Thus, the Shell and Utilities volume of POSIX.1-2017 says that the shell
128258 ``may'' exit in this case. This puts a small burden on the programmer, who has to test for
128259 successful completion following a command if it is important that the next command not be
128260 executed if the previous command was not found. If it is important for the command to have
128261 been found, it was probably also important for it to complete successfully. The test for successful
128262 completion would not need to change.

128263 Historically, shells have returned an exit status of $128+n$, where n represents the signal number.
128264 Since signal numbers are not standardized, there is no portable way to determine which signal
128265 caused the termination. Also, it is possible for a command to exit with a status in the same range
128266 of numbers that the shell would use to report that the command was terminated by a signal.
128267 Implementations are encouraged to choose exit values greater than 256 to indicate programs that
128268 terminate by a signal so that the exit status cannot be confused with an exit status generated by a
128269 normal termination.

128270 Historical shells make the distinction between ``utility not found'' and ``utility found but cannot
128271 execute'' in their error messages. By specifying two seldomly used exit status values for these
128272 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this
128273 distinction without having to parse an error message that would probably change from locale to
128274 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of
128275 POSIX.1-2017 have also been specified to use this convention.

128276 When a command fails during word expansion or redirection, most historical implementations
128277 exit with a status of 1. However, there was some sentiment that this value should probably be
128278 much higher so that an application could distinguish this case from the more normal exit status
128279 values. Thus, the language ``greater than zero'' was selected to allow either method to be
128280 implemented.

128281 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0022 [717] is applied.

128282 **C.2.9 Shell Commands**

128283 A description of an “empty command” was removed from an early proposal because it is only
 128284 relevant in the cases of *sh -c ""*, *system("")*, or an empty shell-script file (such as the
 128285 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell
 128286 and Utilities volume of POSIX.1-2017, it falls into the silently unspecified category of behavior
 128287 where implementations can continue to operate as they have historically, but conforming
 128288 applications do not construct empty commands. (However, note that *sh* does explicitly state an
 128289 exit status for an empty string or file.) In an interactive session or a script with other commands,
 128290 extra <newline> or <semicolon> characters, such as:

```
128291 $ false
128292 $
128293 $ echo $?
128294 1
```

128295 would not qualify as the empty command described here because they would be consumed by
 128296 other parts of the grammar.

128297 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0023 [473] is applied.

128298 **C.2.9.1 Simple Commands**

128299 The enumerated list is used only when the command is actually going to be executed. For
 128300 example, in:

```
128301 true || $foo *
```

128302 no expansions are performed.

128303 The following example illustrates both how a variable assignment without a command name
 128304 affects the current execution environment, and how an assignment with a command name only
 128305 affects the execution environment of the command:

```
128306 $ x=red
128307 $ echo $x
128308 red
128309 $ export x
128310 $ sh -c 'echo $x'
128311 red
128312 $ x=blue sh -c 'echo $x'
128313 blue
128314 $ echo $x
128315 red
```

128316 This next example illustrates that redirections without a command name are still performed:

```
128317 $ ls foo
128318 ls: foo: no such file or directory
128319 $ > foo
128320 $ ls foo
128321 foo
```

128322 A command without a command name, but one that includes a command substitution, has an
 128323 exit status of the last command substitution that the shell performed. For example:

```
128324 if      x=$(command)
128325 then    ...
128326 fi
```

128327 An example of redirections without a command name being performed in a subshell shows that
 128328 the here-document does not disrupt the standard input of the **while** loop:

```
128329 IFS=:
128330 while read a b
128331 do     echo $a
128332       <<-eof
128333       Hello
128334       eof
128335 done </etc/passwd
```

128336 Following are examples of commands without command names in AND-OR lists:

```
128337 > foo || {
128338     echo "error: foo cannot be created" >&2
128339     exit 1
128340 }
128341 # set saved if /vmunix.save exists
128342 test -f /vmunix.save && saved=1
```

128343 Command substitution and redirections without command names both occur in subshells, but
 128344 they are not necessarily the same ones. For example, in:

```
128345 exec 3> file
128346 var=$(echo foo >&3) 3>&1
```

128347 it is unspecified whether **foo** is echoed to the file or to standard output.

128348 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0021 [255] is applied.

128349 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0024 [654] is applied.

128350 **Command Search and Execution**

128351 This description requires that the shell can execute shell scripts directly, even if the underlying
 128352 system does not support the common "#!" interpreter convention. That is, if file **foo** contains
 128353 shell commands and is executable, the following executes **foo**:

```
128354 ./foo
```

128355 The command search shown here does not match all historical implementations. A more typical
 128356 sequence has been:

128357 Any built-in (special or regular)

128358 Functions

128359 Path search for executable files

128360 But there are problems with this sequence. Since the programmer has no idea in advance which
 128361 utilities might have been built into the shell, a function cannot be used to override portably a
 128362 utility of the same name. (For example, a function named *cd* cannot be written for many
 128363 historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only
 128364 a pathname with a <slash> can be used to ensure a specific executable file is invoked.

128365 After the *execve()* failure described, the shell normally executes the file as a shell script. Some
 128366 implementations, however, attempt to detect whether the file is actually a script and not an
 128367 executable from some other architecture. The method used by the KornShell is allowed by the
 128368 text that indicates non-text files may be bypassed.

128369 The sequence selected for the Shell and Utilities volume of POSIX.1-2017 acknowledges that
 128370 special built-ins cannot be overridden, but gives the programmer full control over which
 128371 versions of other utilities are executed. It provides a means of suppressing function lookup (via
 128372 the *command* utility) for the user's own functions and ensures that any regular built-ins or
 128373 functions provided by the implementation are under the control of the path search. The
 128374 mechanisms for associating built-ins or functions with executable files in the path are not
 128375 specified by the Shell and Utilities volume of POSIX.1-2017, but the wording requires that if
 128376 either is implemented, the application is not able to distinguish a function or built-in from an
 128377 executable (other than in terms of performance, presumably). The implementation ensures that
 128378 all effects specified by the Shell and Utilities volume of POSIX.1-2017 resulting from the
 128379 invocation of the regular built-in or function (interaction with the environment, variables, traps,
 128380 and so on) are identical to those resulting from the invocation of an executable file.

128381 Various historical implementations have used the names in item 1.b. as built-ins or reserved
 128382 words. This standard does not specify their behavior, but their existence means that it is
 128383 important for portable applications to avoid giving functions (or utilities in *PATH*) those names
 128384 because the function (or utility in *PATH*) might not be executed as expected.

128385 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/4 is applied, updating the case where
 128386 *execve()* fails due to an error equivalent to the [ENOEXEC] error.

128387 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0022 [168], XCU/TC1-2008/0023
 128388 [168], XCU/TC1-2008/0024 [168], XCU/TC1-2008/0025 [168], XCU/TC1-2008/0026 [168,430],
 128389 XCU/TC1-2008/0027 [168,430], and XCU/TC1-2008/0028 [173] are applied.

128390 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0025 [935] and XCU/TC2-2008/0026
 128391 [705] are applied.

128392 Examples

128393 Consider three versions of the *ls* utility:

- 128394 1. The application includes a shell function named *ls*.
- 128395 2. The user writes a utility named *ls* and puts it in **/fred/bin**.
- 128396 3. The example implementation provides *ls* as a regular shell built-in that is invoked (either
 128397 by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

128398 If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

	Invocation	Version of <i>ls</i>
128399	<i>ls</i> (from within application script)	(1) function
128400	<i>command ls</i> (from within application script)	(3) built-in
128401	<i>ls</i> (from within makefile called by application)	(3) built-in
128402	<i>system("ls")</i>	(3) built-in
128403	<i>PATH="/fred/bin:\$PATH" ls</i>	(2) user's version
128404		

128405 C.2.9.2 Pipelines

128406 Because pipeline assignment of standard input or standard output or both takes place before
128407 redirection, it can be modified by redirection. For example:

```
128408 $ command1 2>&1 | command2
```

128409 sends both the standard output and standard error of *command1* to the standard input of
128410 *command2*.

128411 The reserved word **!** allows more flexible testing using AND and OR lists. The behavior of **!** is
128412 unspecified because in the Korn Shell this introduces a negated pathname expansion. Portable
128413 applications need to separate the **!** and **(** to ensure the command is treated as a negated subshell.

128414 It was suggested that it would be better to return a non-zero value if any command in the
128415 pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values).
128416 However, the choice of the last-specified command semantics are historical practice and would
128417 cause applications to break if changed. An example of historical behavior:

```
128418 $ sleep 5 | (exit 4)
```

```
128419 $ echo $?
```

```
128420 4
```

```
128421 $ (exit 4) | sleep 5
```

```
128422 $ echo $?
```

```
128423 0
```

128424 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0029 [205] is applied.

128425 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0027 [521] is applied.

128426 **Exit Status**

128427 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0030 [52] is applied.

128428 C.2.9.3 Lists

128429 The equal precedence of "&&" and "||" is historical practice. The standard developers
128430 evaluated the model used more frequently in high-level programming languages, such as C, to
128431 allow the shell logical operators to be used for complex expressions in an unambiguous way, but
128432 they could not allow historical scripts to break in the subtle way unequal precedence might
128433 cause. Some arguments were posed concerning the "{}" or "()" groupings that are required
128434 historically. There are some disadvantages to these groupings:

128435 The "()" can be expensive, as they spawn other processes on some implementations. This
128436 performance concern is primarily an implementation issue.

128437 The "{}" braces are not operators (they are reserved words) and require a trailing
128438 <space> after each '{', and a <semicolon> before each '}'. Most programmers (and
128439 certainly interactive users) have avoided braces as grouping constructs because of the
128440 problematic syntax required. Braces were not changed to operators because that would
128441 generate compatibility issues even greater than the precedence question; braces appear
128442 outside the context of a keyword in many shell scripts.

128443 IEEE PASC Interpretation 1003.2 #204 is applied, clarifying that the operators "&&" and "||"
128444 are evaluated with left associativity.

128445 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0031 [45] and XCU/TC1-2008/0032
128446 [45] are applied.

128447 Asynchronous Lists

128448 The grammar treats a construct such as:

```
128449 foo & bar & bam &
```

128450 as one “asynchronous list”, but since the status of each element is tracked by the shell, the term
128451 “element of an asynchronous list” was introduced to identify just one of the **foo**, **bar**, or **bam**
128452 portions of the overall list.

128453 Unless the implementation has an internal limit, such as {CHILD_MAX}, on the retained process
128454 IDs, it would require unbounded memory for the following example:

```
128455 while true  
128456 do     foo & echo $!  
128457 done
```

128458 The treatment of the signals SIGINT and SIGQUIT with asynchronous lists is described in XCU
128459 [Section 2.11](#) (on page 2381).

128460 Since the connection of the input to the equivalent of **/dev/null** is considered to occur before
128461 redirections, the following script would produce no output:

```
128462 exec < /etc/passwd  
128463 cat <&0 &  
128464 wait
```

128465 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0028 [760] is applied.

128466 Sequential Lists

128467 There is no additional rationale provided for this section.

128468 AND Lists

128469 There is no additional rationale provided for this section.

128470 OR Lists

128471 There is no additional rationale provided for this section.

128472 C.2.9.4 Compound Commands

128473 **Grouping Commands**

128474 The semicolon shown in `{compound-list;}` is an example of a control operator delimiting the }
 128475 reserved word. Other delimiters are possible, as shown in XCU Section 2.10 (on page 2375);
 128476 <newline> is frequently used.

128477 A proposal was made to use the <do-done> construct in all cases where command grouping in
 128478 the current process environment is performed, identifying it as a construct for the grouping
 128479 commands, as well as for shell functions. This was not included because the shell already has a
 128480 grouping construct for this purpose ("`{}`"), and changing it would have been counter-
 128481 productive.

128482 The requirement for conforming applications to separate two leading ' (' characters with white
 128483 space if a grouping command would be parsed as an arithmetic expansion if preceded by a '\$'
 128484 is to allow shells which implement the "`((arithmetic expression))`" extension to
 128485 apply the same disambiguation rules consistently to `$(...)` and `((...))`. See Section
 128486 C.2.6.3 (on page 3730).

128487 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0033 [217] is applied.

128488 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] is applied.

128489 **For Loop**

128490 The format is shown with generous usage of <newline> characters. See the grammar in XCU
 128491 Section 2.10 (on page 2375) for a precise description of where <newline> and <semicolon>
 128492 characters can be interchanged.

128493 Some historical implementations support '{' and '}' as substitutes for **do** and **done**. The
 128494 standard developers chose to omit them, even as an obsolescent feature. (Note that these
 128495 substitutes were only for the **for** command; the **while** and **until** commands could not use them
 128496 historically because they are followed by compound-lists that may contain "{...}" grouping
 128497 commands themselves.)

128498 The reserved word pair **do ... done** was selected rather than **do ... od** (which would have
 128499 matched the spirit of **if ... fi** and **case ... esac**) because *od* is already the name of a standard
 128500 utility.

128501 PASC Interpretation 1003.2 #169 has been applied changing the grammar.

128502 **Case Conditional Construct**

128503 An optional <left-parenthesis> before *pattern* was added to allow numerous historical KornShell
 128504 scripts to conform. At one time, using the leading parenthesis was required if the **case** statement
 128505 was to be embedded within a "`$()`" command substitution; this is no longer the case with the
 128506 POSIX shell. Nevertheless, many historical scripts use the <left-parenthesis>, if only because it
 128507 makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple
 128508 implementation change that is upwards-compatible for all scripts.

128509 Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to
 128510 the next pattern action list. This was rejected as being nonexistent practice. An interesting
 128511 undocumented feature of the KornShell is that using "`;&`" instead of "`;;`" as a terminator
 128512 causes the exact opposite behavior—the flow of control continues with the next *compound-list*.

128513 The pattern '`*`', given as the last pattern in a **case** construct, is equivalent to the default case in
 128514 a C-language **switch** statement.

128515 The grammar shows that reserved words can be used as patterns, even if one is the first word on

128516 a line. Obviously, the reserved word **esac** cannot be used in this manner.
 128517 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] is applied.

128518 **If Conditional Construct**

128519 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2375).

128520 **While Loop**

128521 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2375).

128522 **Until Loop**

128523 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2375).

128524 C.2.9.5 *Function Definition Command*

128525 The description of functions in an early proposal was based on the notion that functions should
 128526 behave like miniature shell scripts; that is, except for sharing variables, most elements of an
 128527 execution environment should behave as if they were a new execution environment, and
 128528 changes to these should be local to the function. For example, traps and options should be reset
 128529 on entry to the function, and any changes to them do not affect the traps or options of the caller.
 128530 There were numerous objections to this basic idea, and the opponents asserted that functions
 128531 were intended to be a convenient mechanism for grouping common commands that were to be
 128532 executed in the current execution environment, similar to the execution of the *dot* special
 128533 built `-in`.

128534 It was also pointed out that the functions described in that early proposal did not provide a local
 128535 scope for everything a new shell script would, such as the current working directory, or *umask*,
 128536 but instead provided a local scope for only a few select properties. The basic argument was that
 128537 if a local scope is needed for the execution environment, the mechanism already existed: the
 128538 application can put the commands in a new shell script and call that script. All historical shells
 128539 that implemented functions, other than the KornShell, have implemented functions that operate
 128540 in the current execution environment. Because of this, traps and options have a global scope
 128541 within a shell script. Local variables within a function were considered and included in another
 128542 early proposal (controlled by the special built-in *local*), but were removed because they do not fit
 128543 the simple model developed for functions and because there was some opposition to adding yet
 128544 another new special built-in that was not part of historical practice. Implementations should
 128545 reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable
 128546 mechanism is adopted in a future version of this standard.

128547 A separate issue from the execution environment of a function is the availability of that function
 128548 to child shells. A few objectors maintained that just as a variable can be shared with child shells
 128549 by exporting it, so should a function. In early proposals, the *export* command therefore had a `-f`
 128550 flag for exporting functions. Functions that were exported were to be put into the environment
 128551 as *name()*=*value* pairs, and upon invocation, the shell would scan the environment for these and
 128552 automatically define these functions. This facility was strongly opposed and was omitted. Some
 128553 of the arguments against exportable functions were as follows:

128554 There was little historical practice. The Ninth Edition shell provided them, but there was
 128555 controversy over how well it worked.

128556 There are numerous security problems associated with functions appearing in the
 128557 environment of a user and overriding standard utilities or the utilities owned by the
 128558 application.

128559 There was controversy over requiring *make* to import functions, where it has historically
 128560 used an *exec* function for many of its command line executions.

128561 Functions can be big and the environment is of a limited size. (The counter-argument was
 128562 that functions are no different from variables in terms of size: there can be big ones, and
 128563 there can be small ones ‡and just as one does not export huge variables, one does not
 128564 export huge functions. However, this might not apply to the average shell-function writer,
 128565 who typically writes much larger functions than variables.)

128566 As far as can be determined, the functions in the Shell and Utilities volume of POSIX.1-2017
 128567 match those in System V. Earlier versions of the KornShell had two methods of defining
 128568 functions:

```
128569           function fname { compound-list }
```

128570 and:

```
128571           fname() { compound-list }
```

128572 The latter used the same definition as the Shell and Utilities volume of POSIX.1-2017, but
 128573 differed in semantics, as described previously. The current edition of the KornShell aligns the
 128574 latter syntax with the Shell and Utilities volume of POSIX.1-2017 and keeps the former as is.

128575 The name space for functions is limited to that of a *name* because of historical practice.
 128576 Complications in defining the syntactic rules for the function definition command and in
 128577 dealing with known extensions such as the "@()" usage in the KornShell prevented the name
 128578 space from being widened to a *word*. Using functions to support synonyms such as the "!!"
 128579 and '%' usage in the C shell is thus disallowed to conforming applications, but acceptable as an
 128580 extension. For interactive users, the aliasing facilities in the Shell and Utilities volume of
 128581 POSIX.1-2017 should be adequate for this purpose. It is recognized that the name space for
 128582 utilities in the file system is wider than that currently supported for functions, if the portable
 128583 filename character set guidelines are ignored, but it did not seem useful to mandate extensions
 128584 in systems for so little benefit to conforming applications.

128585 The "()" in the function definition command consists of two operators. Therefore, intermixing
 128586 <blank> characters with the *fname*, '(', and ')' is allowed, but unnecessary.

128587 An example of how a function definition can be used wherever a simple command is allowed:

```
128588           # If variable i is equal to "yes",
128589           # define function foo to be ls -l
128590           #
128591           [ "$i" = yes ] && foo() {
128592                 ls -l
128593           }
```

128594 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0034 [383] and XCU/TC1-2008/0035
 128595 [214] are applied.

128596 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0029 [473] and XCU/TC2-2008/0030
 128597 [654] are applied.

128598 **C.2.10 Shell Grammar**

128599 There are several subtle aspects of this grammar where conventional usage implies rules about
128600 the grammar that in fact are not true.

128601 For *compound_list*, only the forms that end in a *separator* allow a reserved word to be recognized,
128602 so usually only a *separator* can be used where a compound list precedes a reserved word (such as
128603 **Then, Else, Do, and Rbrace**). Explicitly requiring a separator would disallow such valid (if rare)
128604 statements as:

```
128605 if (false) then (echo x) else (echo y) fi
```

128606 See the Note under special grammar rule (1).

128607 Concerning the third sentence of rule (1) (“Also, if the parser ...”):

128608 This sentence applies rather narrowly: when a compound list is terminated by some clear
128609 delimiter (such as the closing **fi** of an inner **if_clause**) then it would apply; where the
128610 compound list might continue (as in after a **;**), rule (7a) (and consequently the first
128611 sentence of rule (1)) would apply. In many instances the two conditions are identical, but
128612 this part of rule (1) does not give license to treating a **WORD** as a reserved word unless it
128613 is in a place where a reserved word has to appear.

128614 The statement is equivalent to requiring that when the LR(1) lookahead set contains
128615 exactly one reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the
128616 theoretical concepts, not to any real parser generator.)

128617 For example, in the construct below, and when the parser is at the point marked with **^**,
128618 the only next legal token is **then** (this follows directly from the grammar rules):

```
128619 if if...fi then ... fi  
128620 ^
```

128621 At that point, the **then** must be recognized as a reserved word.

128622 (Depending on the parser generator actually used, “extra” reserved words may be in some
128623 lookahead sets. It does not really matter if they are recognized, or even if any possible
128624 reserved word is recognized in that state, because if it is recognized and is not in the
128625 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if
128626 some other reserved word (for example, **while**) is also recognized, an error occurs later.

128627 This is approximately equivalent to saying that reserved words are recognized after other
128628 reserved words (because it is after a reserved word that this condition occurs), but avoids
128629 the “except for ...” list that would be required for **case**, **for**, and so on. (Reserved words
128630 are of course recognized anywhere a *simple_command* can appear, as well. Other rules take
128631 care of the special cases of non-recognition, such as rule (4) for **case** statements.)

128632 Note that the body of here-documents are handled by token recognition (see XCU [Section 2.3](#), on
128633 page 2347) and do not appear in the grammar directly. (However, the here-document I/O
128634 redirection operator is handled as part of the grammar.)

128635 **C.2.10.1 Shell Grammar Lexical Conventions**

128636 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0031 [648] and XCU/TC2-2008/0032
128637 [574,646] are applied.

128638 C.2.10.2 Shell Grammar Rules

128639 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0036 [44] is applied.

128640 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0033 [643,839], XCU/TC2-2008/0034
128641 [643], XCU/TC2-2008/0035 [648], XCU/TC2-2008/0036 [736], XCU/TC2-2008/0037 [737],
128642 XCU/TC2-2008/0038 [581], and XCU/TC2-2008/0039 [735] are applied.

128643 C.2.11 Signals and Error Handling

128644 Historically, some shell implementations silently ignored attempts to use *trap* to set SIGINT or
128645 SIGQUIT to the default action or to set a trap for them after they have been set to be ignored by
128646 the shell when it executes an asynchronous subshell (and job control is disabled). This behavior
128647 is not conforming. For example, if a shell script containing the following line is run in the
128648 foreground at a terminal:

128649

```
(trap - INT; exec sleep 10) & wait
```

128650 and is then terminated by typing the interrupt character, this standard requires that the *sleep*
128651 command is terminated by the SIGINT signal.

128652 SD5-XCU-ERN-93 is applied, updating the first paragraph of XCU [Section 2.11](#) (on page 2381).

128653 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0040 [750] is applied.

128654 C.2.12 Shell Execution Environment

128655 Some implementations have implemented the last stage of a pipeline in the current environment
128656 so that commands such as:

128657

```
command | read foo
```

128658 set variable **foo** in the current environment. This extension is allowed, but not required;
128659 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but
128660 not depend on it.

128661 In early proposals, the description of execution environment failed to mention that each
128662 command in a multiple command pipeline could be in a subshell execution environment. For
128663 compatibility with some historical shells, the wording was phrased to allow an implementation
128664 to place any or all commands of a pipeline in the current environment. However, this means that
128665 a POSIX application must assume each command is in a subshell environment, but not depend
128666 on it.

128667 The wording about shell scripts is meant to convey the fact that describing “trap actions” can
128668 only be understood in the context of the shell command language. Outside of this context, such
128669 as in a C-language program, signals are the operative condition, not traps.

128670 POSIX.1-2008, Technical Corrigendum 1, XCU/TC1-2008/0037 [238] is applied.

128671 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0041 [706] is applied.

128672 **C.2.13 Pattern Matching Notation**

128673 Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is
 128674 generally used for the manipulation of filenames, which are relatively simple collections of
 128675 characters, while the latter is generally used to manipulate arbitrary text strings of potentially
 128676 greater complexity. However, some of the basic concepts are the same, so this section points
 128677 liberally to the detailed descriptions in XBD [Chapter 9](#) (on page 181).

128678 *C.2.13.1 Patterns Matching a Single Character*

128679 Both quoting and escaping are described here because pattern matching must work in three
 128680 separate circumstances:

128681 1. Calling directly upon the shell, such as in pathname expansion or in a **case** statement. All
 128682 of the following match the string or file **abc**:

128683 `abc "abc" a"b" c a\bc a[b]c a["b"]c a[\b]c a["\b"]c a?c a*c`

128684 The following do not:

128685 `"a?c" a*c a\b]c`

128686 2. Calling a utility or function without going through a shell, as described for *find* and the
 128687 *fnmatch()* function defined in the System Interfaces volume of POSIX.1-2017.

128688 3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case,
 128689 shell quote removal is performed before the utility sees the argument. For example, in:

128690 `find /bin -name "e\c[\h]o" -print`

128691 after quote removal, the `<backslash>` characters are presented to *find* and it treats them as
 128692 escape characters. Both precede ordinary characters, so the *c* and *h* represent themselves
 128693 and *echo* would be found on many historical systems (that have it in **/bin**). To find a
 128694 filename that contained shell special characters or pattern characters, both quoting and
 128695 escaping are required, such as:

128696 `pax -r ... "*a\(\?"`

128697 to extract a filename ending with `"a(?"`.

128698 Conforming applications are required to quote or escape the shell special characters (sometimes
 128699 called metacharacters). If used without this protection, syntax errors can result or
 128700 implementation extensions can be triggered. For example, the KornShell supports a series of
 128701 extensions based on parentheses in patterns.

128702 The restriction on a `<circumflex>` in a bracket expression is to allow implementations that
 128703 support pattern matching using the `<circumflex>` as the negation character in addition to the
 128704 `<exclamation-mark>`. A conforming application must use something like `"[\^!]"` to match
 128705 either character.

128706 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0042 [806] is applied.

128707 *C.2.13.2 Patterns Matching Multiple Characters*

128708 Since each `<asterisk>` matches zero or more occurrences, the patterns `"a*b"` and `"a**b"` have
 128709 identical functionality.

128710 **Examples**

- 128711 a[bc] Matches the strings "ab" and "ac".
- 128712 a*d Matches the strings "ad", "abd", and "abcd", but not the string "abc".
- 128713 a*d* Matches the strings "ad", "abcd", "abcdef", "aaaad", and "adddd".
- 128714 *a*d Matches the strings "ad", "abcd", "efabcd", "aaaad", and "adddd".

128715 C.2.13.3 *Patterns Used for Filename Expansion*

128716 The caveat about a <slash> within a bracket expression is derived from historical practice. The
 128717 pattern "a[b/c]d" does not match such pathnames as **abd** or **a/d**. On some implementations
 128718 (including those conforming to the Single UNIX Specification), it matched a pathname of
 128719 literally "a[b/c]d". On other systems, it produced an undefined condition (an unescaped '['
 128720 used outside a bracket expression). In this version, the XSI behavior is now required.

128721 Filenames beginning with a <period> historically have been specially protected from view on
 128722 UNIX systems. A proposal to allow an explicit <period> in a bracket expression to match a
 128723 leading <period> was considered; it is allowed as an implementation extension, but a
 128724 conforming application cannot make use of it. If this extension becomes popular in the future, it
 128725 will be considered for a future version of the Shell and Utilities volume of POSIX.1-2017.

128726 Historical systems have varied in their permissions requirements. To match **f*/bar** has required
 128727 read permissions on the **f*** directories in the System V shell, but the Shell and Utilities volume of
 128728 POSIX.1-2017, the C shell, and KornShell require only search permissions.

128729 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0043 [963] is applied.

128730 **C.2.14 Special Built-In Utilities**

- 128731 See the RATIONALE sections on the individual reference pages.
- 128732 POSIX.1-2008, Technical Corrigendum 2, XCU/TC2-2008/0044 [882] and XCU/TC2-2008/0045
 128733 [654] are applied.

128734 **C.3 Batch Environment Services and Utilities**128735 **Scope of the Batch Environment Services and Utilities Option**

128736 This section summarizes the deliberations of the IEEE P1003.15 (Batch Environment) working
 128737 group in the development of the Batch Environment Services and Utilities option, which covers
 128738 a set of services and utilities defining a batch processing system.

128739 This informative section contains historical information concerning the contents of the
 128740 amendment and describes why features were included or discarded by the working group.

128741 History of Batch Systems

128742 The supercomputing technical committee began as a “Birds Of a Feather” (BOF) at the January
128743 1987 Usenix meeting. There was enough general interest to form a supercomputing attachment
128744 to the /usr/group working groups. Several subgroups rapidly formed. Of those subgroups, the
128745 batch group was the most ambitious. The first early meetings were spent evaluating user needs
128746 and existing batch implementations.

128747 To evaluate user needs, individuals from the supercomputing community came and presented
128748 their needs. Common requests were flexibility, interoperability, control of resources, and ease-of-
128749 use. Backward-compatibility was not an issue. The working group then evaluated some existing
128750 systems. The following different systems were evaluated:

128751 PROD

128752 Convex Distributed Batch

128753 NQS

128754 CTSS

128755 MDQS from Ballistics Research Laboratory (BRL)

128756 Finally, NQS was chosen as a model because it satisfied not only the most user requirements, but
128757 because it was public domain, already implemented on a variety of hardware platforms, and
128758 network-based.

128759 Historical Implementations of Batch Systems

128760 Deferred processing of work under the control of a scheduler has been a feature of most
128761 proprietary operating systems from the earliest days of multi-user systems in order to maximize
128762 utilization of the computer.

128763 The arrival of UNIX systems proved to be a dilemma to many hardware providers and users
128764 because it did not include the sophisticated batch facilities offered by the proprietary systems.
128765 This omission was rectified in 1986 by NASA Ames Research Center who developed the
128766 Network Queuing System (NQS) as a portable UNIX application that allowed the routing and
128767 processing of batch “jobs” in a network. To encourage its usage, the product was later put into
128768 the public domain. It was promptly picked up by UNIX hardware providers, and ported and
128769 developed for their respective hardware and UNIX implementations.

128770 Many major vendors, who traditionally offer a batch-dominated environment, ported the
128771 public-domain product to their systems, customized it to support the capabilities of their
128772 systems, and added many customer-requested features.

128773 Due to the strong hardware provider and customer acceptance of NQS, it was decided to use
128774 NQS as the basis for the POSIX Batch Environment amendment in 1987. Other batch systems
128775 considered at the time included CTSS, MDQS (a forerunner of NQS from the Ballistics Research
128776 Laboratory), and PROD (a Los Alamos Labs development). None were thought to have both the
128777 functionality and acceptability of NQS.

128778 **NQS Differences from the *at* utility**

128779 The base standard *at* and *batch* utilities are not sufficient to meet the batch processing needs in a
 128780 supercomputing environment and additional functionality in the areas of resource management,
 128781 job scheduling, system management, and control of output is required.

128782 **Batch Environment Services and Utilities Option Definitions**

128783 The concept of a batch job is closely related to a session with a session leader. The main
 128784 difference is that a batch job does not have a controlling terminal. There has been much debate
 128785 over whether to use the term “request” or “job”. Job was the final choice because of the
 128786 historical use of this term in the batch environment.

128787 The current definition for job identifiers is not sufficient with the model of destinations. The
 128788 current definition is:

128789 `sequence_number.originating_host`

128790 Using the model of destination, a host may include multiple batch nodes, the location of which
 128791 is identified uniquely by a name or directory service. If the current definition is used, batch
 128792 nodes running on the same host would have to coordinate their use of sequence numbers, as
 128793 sequence numbers are assigned by the originating host. The alternative is to use the originating
 128794 batch node name instead of the originating host name.

128795 The reasons for wishing to run more than one batch system per host could be the following.

128796 A test and production batch system are maintained on a single host. This is most likely in a
 128797 development facility, but could also arise when a site is moving from one version to another. The
 128798 new batch system could be installed as a test version that is completely separate from the
 128799 production batch system, so that problems can be isolated to the test system. Requiring the batch
 128800 nodes to coordinate their use of sequence numbers creates a dependency between the two
 128801 nodes, and that defeats the purpose of running two nodes.

128802 A site has multiple departments using a single host, with different management policies. An
 128803 example of contention might be in job selection algorithms. One group might want a FIFO type
 128804 of selection, while another group wishes to use a more complex algorithm based on resource
 128805 availability. Again, requiring the batch nodes to coordinate is an unnecessary binding.

128806 The proposal eventually accepted was to replace originating host with originating batch node.
 128807 This supplies sufficient granularity to ensure unique job identifiers. If more than one batch node
 128808 is on a particular host, they each have their own unique name.

128809 The queue portion of a destination is not part of the job identifier as these are not required to be
 128810 unique between batch nodes. For instance, two batch nodes may both have queues called small,
 128811 medium, and large. It is only the batch node name that is uniquely identifiable throughout the
 128812 batch system. The queue name has no additional function in this context.

128813 Assume there are three batch nodes, each of which has its own name server. On batch node one,
 128814 there are no queues. On batch node two, there are fifty queues. On batch node three, there are
 128815 forty queues. The system administrator for batch node one does not have to configure queues,
 128816 because there are none implemented. However, if a user wishes to send a job to either batch
 128817 node two or three, the system administrator for batch node one must configure a destination
 128818 that maps to the appropriate batch node and queue. If every queue is to be made accessible from
 128819 batch node one, the system administrator has to configure ninety destinations.

128820 To avoid requiring this, there should be a mechanism to allow a user to separate the destination
 128821 into a batch node name and a queue name. Then, an implementation that is configured to get to
 128822 all the batch nodes does not need any more configuration to allow a user to get to all of the
 128823 queues on all of the batch nodes. The node name is used to locate the batch node, while the

- 128824 queue name is sent unchanged to that batch node.
- 128825 The following are requirements that a destination identifier must be capable of providing:
- 128826 The ability to direct a job to a queue in a particular batch node.
- 128827 The ability to direct a job to a particular batch node.
- 128828 The ability to group at a higher level than just one queue. This includes grouping similar queues across multiple batch nodes (this is a pipe queue).
- 128829
- 128830 The ability to group batch nodes. This allows a user to submit a job to a group name with no knowledge of the batch node configuration. This also provides aliasing as a special case. Aliasing is a group containing only one batch node name. The group name is the alias.
- 128831
- 128832
- 128833
- 128834 In addition, the administrator has the following requirements:
- 128835 The ability to control access to the queues.
- 128836 The ability to control access to the batch nodes.
- 128837 The ability to control access to groups of queues (pipe queues).
- 128838 The ability to configure retry time intervals and durations.
- 128839 The requirements of the user are met by destination as explained in the following.
- 128840 The user has the ability to specify a queue name, which is known only to the batch node specified. There is no configuration of these queues required on the submitting node.
- 128841
- 128842 The user has the ability to specify a batch node whose name is network-unique. The configuration required is that the batch node be defined as an application, just as other applications such as FTP are configured.
- 128843
- 128844
- 128845 Once a job reaches a queue, it can again become a user of the batch system. The batch node can choose to send the job to another batch node or queue or both. In other words, the routing is at an application level, and it is up to the batch system to choose where the job will be sent. Configuration is up to the batch node where the queue resides. This provides grouping of queues across batch nodes or within a batch node. The user submits the job to a queue, which by definition routes the job to other queues or nodes or both.
- 128846
- 128847
- 128848
- 128849
- 128850
- 128851 A node name may be given to a naming service, which returns multiple addresses as opposed to just one. This provides grouping at a batch node level. This is a local issue, meaning that the batch node must choose only one of these addresses. The list of addresses is not sent with the job, and once the job is accepted on another node, there is no connection between the list and the job. The requirements of the administrator are met by destination as explained in the following.
- 128852
- 128853
- 128854
- 128855
- 128856 The control of queues is a batch system issue, and will be done using the batch administrative utilities.
- 128857
- 128858 The control of nodes is a network issue, and will be done through whatever network facilities are available.
- 128859
- 128860 The control of access to groups of queues (pipe queues) is covered by the control of any other queue. The fact that the job may then be sent to another destination is not relevant.
- 128861
- 128862 The propagation of a job across more than one point-to-point connection was dropped because of its complexity and because all of the issues arising from this capability could not be resolved. It could be provided as additional functionality at some time in the future.
- 128863
- 128864
- 128865 The addition of *network* as a defined term was done to clarify the difference between a network of batch nodes as opposed to a network of hosts. A network of batch nodes is referred to as a
- 128866

128867 batch system. The network refers to the actual host configuration. A single host may have
128868 multiple batch nodes.

128869 In the absence of a standard network naming convention, this option establishes its own
128870 convention for the sake of consistency and expediency. This is subject to change, should a future
128871 working group develop a standard naming convention for network pathnames.

128872 C.3.1 Batch General Concepts

128873 During the development of the Batch Environment Services and Utilities option, a number of
128874 topics were discussed at length which influenced the wording of the normative text but could
128875 not be included in the final text. The following items are some of the most significant terms and
128876 concepts of those discussed:

128877 Small and Consistent Command Set

128878 Often, conventional utilities from UNIX systems have a very complicated utility syntax
128879 and usage. This can often result in confusion and errors when trying to use them. The
128880 Batch Environment Services and Utilities option utility set, on the other hand, has been
128881 paired to a small set of robust utilities with an orthogonal calling sequence.

128882 Checkpoint/Restart

128883 This feature permits an already executing process to checkpoint or save its contents. Some
128884 implementations permit this at both the batch utility level (for example, checkpointing this
128885 job upon its abnormal termination) or from within the job itself via a system call. Support
128886 of checkpoint/restart is optional. A conscious, careful effort was made to make the *qsub*
128887 utility consistently refer to checkpoint/restart as optional functionality.

128888 Rerunability

128889 When a user submits a job for batch processing, they can designate it “rerunnable” in that
128890 it will automatically resume execution from the start of the job if the machine on which it
128891 was executing crashes for some reason. The decision on whether the job will be rerun or
128892 not is entirely up to the submitter of the job and no decisions will be made within the batch
128893 system. A job that is rerunnable and has been submitted with the proper
128894 checkpoint/restart switch will first be checkpointed and execution begun from that point.
128895 Furthermore, use of the implementation-defined checkpoint/restart feature will not be
128896 defined in this context.

128897 Error Codes

128898 All utilities exit with error status zero (0) if successful, one (1) if a user error occurred, and
128899 two (2) for an internal Batch Environment Services and Utilities option error.

128900 Level of Portability

128901 Portability is specified at both the user, operator, and administrator levels. A conforming
128902 batch implementation prevents identical functionality and behavior at all these levels.
128903 Additionally, portable batch shell scripts with embedded Batch Environment Services and
128904 Utilities option utilities add an additional level of portability.

128905 Resource Specification

128906 A small set of globally understood resources, such as memory and CPU time, is specified.
128907 All conforming batch implementations are able to process them in a manner consistent
128908 with the yet-to-be-developed resource management model. Resources not in this
128909 amendment set are ignored and passed along as part of the argument stream of the utility.

- 128910 Queue Position
- 128911 Queue position is the place a job occupies in a queue. It is dependent on a variety of factors
128912 such as submission time and priority. Since priority may be affected by the implementation
128913 of fair share scheduling, the definition of queue position is implementation-defined.
- 128914 Queue ID
- 128915 A numerical queue ID is an external requirement for purposes of accounting. The
128916 identification number was chosen over queue name for processing convenience.
- 128917 Job ID
- 128918 A common notion of “jobs” is a collection of processes whose process group cannot be
128919 altered and is used for resource management and accounting. This concept is
128920 implementation-defined and, as such, has been omitted from the batch amendment.
- 128921 Bytes *versus* Words
- 128922 Except for one case, bytes are used as the standard unit for memory size. Furthermore, the
128923 definition of a word varies from machine to machine. Therefore, bytes will be the default
128924 unit of memory size.
- 128925 Regular Expressions
- 128926 The standard definition of regular expressions is much too broad to be used in the batch
128927 utility syntax. All that is needed is a simple concept of “all”; for example, delete all my jobs
128928 from the named queue. For this reason, regular expressions have been eliminated from the
128929 batch amendment.
- 128930 Display Privacy
- 128931 How much data should be displayed locally through functions? Local policy dictates the
128932 amount of privacy. Library functions must be used to create and enforce local policy.
128933 Network and local *qstats* must reflect the policy of the server machine.
- 128934 Remote Host Naming Convention
- 128935 It was decided that host names would be a maximum of 255 characters in length, with at
128936 most 15 characters being shown in displays. The 255 character limit was chosen because it
128937 is consistent with BSD. The 15-character limit was an arbitrary decision.
- 128938 Network Administration
- 128939 Network administration is important, but is outside the scope of the batch amendment.
128940 Network administration could be done with *rsh*. However, authentication becomes two-
128941 sided.
- 128942 Network Administration Philosophy
- 128943 Keep it simple. Centralized management should be possible. For example, Los Alamos
128944 needs a dumb set of CPUs to be managed by a central system *versus* several
128945 independently-managed systems as is the general case for the Batch Environment Services
128946 and Utilities option.
- 128947 Operator Utility Defaults (that is, Default Host, User, Account, and so on)
- 128948 It was decided that usability would override orthogonality and syntactic consistency.
- 128949 The Batch System Manager and Operator Distinction
- 128950 The distinction between manager and operator is that operators can only control the flow
128951 of jobs. A manager can alter the batch system configuration in addition to job flow. POSIX

128952 makes a distinction between user and system administrator but goes no further. The
128953 concepts of manager and operator privileges fall under local policy. The distinction
128954 between manager and operator is historical in batch environments, and the Batch
128955 Environment Services and Utilities option has continued that distinction.

128956 The Batch System Administrator

128957 An administrator is equivalent to a batch system manager.

128958 C.3.2 Batch Services

128959 This rationale is provided as informative rather than normative text, to avoid placing
128960 requirements on implementors regarding the use of symbolic constants, but at the same time to
128961 give implementors a preferred practice for assigning values to these constants to promote
128962 interoperability.

128963 The *Checkpoint* and *Minimum_Cpu_Interval* attributes induce a variety of behavior depending
128964 upon their values. Some jobs cannot or should not be checkpointed. Other users will simply
128965 need to ensure job continuation across planned downtimes; for example, scheduled preventive
128966 maintenance. For users consuming expensive resources, or for jobs that run longer than the
128967 mean time between failures, however, periodic checkpointing may be essential. However,
128968 system administrators must be able to set minimum checkpoint intervals on a queue-by-queue
128969 basis to guard against, for example, naive users specifying interval values too small on memory-
128970 intensive jobs. Otherwise, system overhead would adversely affect performance.

128971 The use of symbolic constants, such as `NO_CHECKPOINT`, was introduced to lend a degree of
128972 formalism and portability to this option.

128973 Support for checkpointing is optional for servers. However, clients must provide for the `-c`
128974 option, since in a distributed environment the job may run on a server that does provide such
128975 support, even if the host of the client does not support the checkpoint feature.

128976 If the user does not specify the `-c` option, the default action is left unspecified by this option.
128977 Some implementations may wish to do checkpointing by default; others may wish to checkpoint
128978 only under an explicit request from the user.

128979 The *Priority* attribute has been made non-optional. All clients already had been required to
128980 support the `-p` option. The concept of prioritization is common in historical implementations.
128981 The default priority is left to the server to establish.

128982 The *Hold_Types* attribute has been modified to allow for implementation-defined hold types to
128983 be passed to a batch server.

128984 It was the intent of the IEEE P1003.15 working group to mandate the support for the
128985 *Resource_List* attribute in this option by referring to another amendment, specifically the
128986 IEEE P1003.1a draft standard. However, during the development of the IEEE P1003.1a draft
128987 standard this was excluded. As such this requirement has been removed from the normative
128988 text.

128989 The *Shell_Path* attribute has been modified to accept a list of shell paths that are associated with
128990 a host. The name of the attribute has been changed to *Shell_Path_List*.

128991 **C.3.3 Common Behavior for Batch Environment Utilities**

128992 This section was defined to meet the goal of a “Small and Consistent Command Set” for this
128993 option.

128994 **C.4 Utilities**

128995 For the utilities included in POSIX.1-2017, see the RATIONALE sections on the individual
128996 reference pages.

128997 **C.4.1 Utilities Removed in this Version**

128998 None.

128999 **C.4.2 Utilities Removed in the Previous Version**

129000 The following utilities were removed in the previous version of this standard:

129001	<i>calendar</i>	<i>cu</i>	<i>line</i>	<i>pcat</i>	<i>unpack</i>
129002	<i>cancel</i>	<i>dircmp</i>	<i>lint</i>	<i>pg</i>	<i>uulog</i>
129003	<i>cc</i>	<i>dis</i>	<i>lpstat</i>	<i>spell</i>	<i>uuname</i>
129004	<i>col</i>	<i>egrep</i>	<i>mail</i>	<i>sum</i>	<i>uupick</i>
129005	<i>cpio</i>	<i>fgrep</i>	<i>pack</i>	<i>tar</i>	<i>uuto</i>

129006 **C.4.3 Exclusion of Utilities**

129007 The set of utilities contained in POSIX.1-2017 is drawn from the base documents for IEEE Std
129008 1003.2-1992, with one addition: the *c99* utility. This section contains rationale for some of the
129009 deliberations that led to this set of utilities, and why certain utilities were excluded.

129010 Many utilities were evaluated by the standard developers; more historical utilities were
129011 excluded from the base documents for IEEE Std 1003.2-1992 than included. The following list
129012 contains many common UNIX system utilities that were not included as mandatory utilities, in
129013 the User Portability Utilities option, in the XSI option, or in one of the software development
129014 groups. It is logistically difficult for this rationale to distribute correctly the reasons for not
129015 including a utility among the various utility options. Therefore, this section covers the reasons
129016 for all utilities not included in POSIX.1-2017.

129017 This rationale is limited to a discussion of only those utilities actively or indirectly evaluated by
129018 the IEEE Std 1003.2-1992 standard developers, rather than the list of all known UNIX utilities
129019 from all its variants.

129020 *adb* The intent of the various software development utilities was to assist in the
129021 installation (rather than the actual development and debugging) of applications.
129022 This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb*
129023 are very hardware-specific.


129024 *as* Assemblers are hardware-specific and are included implicitly as part of the
129025 compilers in POSIX.1-2017.

129026	<i>banner</i>	The only known use of this command is as part of the <i>lp</i> printer header pages. It was decided that the format of the header is implementation-defined, so this utility is superfluous to application portability.
129027		
129028		
129029	<i>calendar</i>	This reminder service program is not useful to conforming applications.
129030	<i>cancel</i>	The <i>lp</i> (line printer spooling) system specified is the most basic possible and did not need this level of application control.
129031		
129032	<i>chroot</i>	This is primarily of administrative use, requiring superuser privileges.
129033	<i>col</i>	No utilities defined in POSIX.1-2017 produce output requiring such a filter. The <i>nroff</i> text formatter is present on many historical systems and will continue to remain as an extension; <i>col</i> is expected to be shipped by all the systems that ship <i>nroff</i> .
129034		
129035		
129036		
129037	<i>cpio</i>	This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
129038	<i>cpp</i>	This is subsumed by <i>c99</i> .
129039	<i>cu</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
129040		
129041	<i>dc</i>	The functionality of this utility can be provided by the <i>bc</i> utility; <i>bc</i> was selected because it was easier to use and had superior functionality. Although the historical versions of <i>bc</i> are implemented using <i>dc</i> as a base, POSIX.1-2017 prescribes the interface and not the underlying mechanism used to implement it.
129042		
129043		
129044		
129045	<i>dircmp</i>	Although a useful concept, the historical output of this directory comparison program is not suitable for processing in application programs. Also, the <i>diff -r</i> command gives equivalent functionality.
129046		
129047		
129048	<i>dis</i>	Disassemblers are hardware-specific.
129049	<i>emacs</i>	The community of <i>emacs</i> editing enthusiasts was adamant that the full <i>emacs</i> editor not be included in IEEE Std 1003.2-1992 because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original <i>emacs</i> program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include <i>emacs</i> , or included it without supporting it, but there were very few that did not include and support <i>vi</i> .
129050		
129051		
129052		
129053		
129054		
129055		
129056		
129057	<i>ld</i>	This is subsumed by <i>c99</i> .
129058	<i>line</i>	The functionality of <i>line</i> can be provided with <i>read</i> .
129059	<i>lint</i>	This technology is partially subsumed by <i>c99</i> . It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what <i>lint</i> would do in these cases is equally difficult.
129060		
129061		
129062		It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of <i>lint</i> is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require <i>lint</i> to detect all errors of any given class.
129063		
129064		
129065		
129066		
129067		
129068		Additionally, some vendors complained that since many compilers are distributed in a binary form without a <i>lint</i> facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much
129069		
129070		

129071		harder. Rather than being able to build upon a standard compiler component (simply by providing <i>c99</i> as an interface), source to that compiler would most likely need to be modified to provide the <i>lint</i> functionality. This was considered a major burden on system providers for a very small gain to developers (users).
129072		
129073		
129074		
129075	<i>login</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
129076		
129077	<i>lorder</i>	This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.
129078		
129079	<i>lpstat</i>	The <i>lp</i> system specified is the most basic possible and did not need this level of application control.
129080		
129081	<i>mail</i>	This utility was omitted in favor of <i>mailx</i> because there was a considerable functionality overlap between the two.
129082		
129083	<i>mknod</i>	This was omitted in favor of <i>mkfifo</i> , as <i>mknod</i> has too many implementation-defined functions.
129084		
129085	<i>news</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs.
129086		
129087	<i>pack</i>	This compression program was considered inferior to <i>compress</i> .
129088	<i>passwd</i>	This utility was proposed in an early draft of the IEEE Std 1003.2-1992 UPE but met with too many objections to be included. There were various reasons:
129089		
129090		Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect.
129091		
129092		
129093		Even though the text in early drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a <i>passwd</i> utility that would conform to POSIX.1-2017 for all sites using their system.
129094		
129095		
129096		
129097		
129098		
129099		This is really a subject for a system administration working group or a security working group.
129100		
129101	<i>pcat</i>	This compression program was considered inferior to <i>zcat</i> .
129102	<i>pg</i>	This duplicated many of the features of the <i>more</i> pager, which was preferred by the standard developers.
129103		
129104	<i>prof</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
129105		
129106		
129107	RCS	RCS was originally considered as part of a version control utilities portion of the scope. However, this aspect was abandoned by the standard developers. SCCS is now included as an optional part of the XSI option.
129108		
129109		
129110	<i>red</i>	Restricted editor. This was not considered by the standard developers because it never provided the level of security restriction required.
129111		
129112	<i>rsh</i>	Restricted shell. This was not considered by the standard developers because it does not provide the level of security restriction that is implied by historical documentation.
129113		
129114		

129115	<i>sdb</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, some useful aspects of <i>sdb</i> are very hardware-specific.
129116		
129117		
129118		
129119	<i>sdiff</i>	The “side-by-side <i>diff</i> ” utility from System V was omitted because it is used infrequently, and even less so by conforming applications. Despite being in System V, it is not in the SVID or XPG.
129120		
129121		
129122		
129122	<i>shar</i>	Any of the numerous “shell archivers” were excluded because they did not meet the requirement of existing practice.
129123		
129124	<i>shl</i>	This utility is terminal-oriented and is not useful from shell scripts or typical application programs. The job control aspects of the shell command language are generally more useful.
129125		
129126		
129127		
129127	<i>size</i>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
129128		
129129		
129130		
129130	<i>spell</i>	This utility is not useful from shell scripts or typical application programs. The <i>spell</i> utility was considered, but was omitted because there is no known technology that can be used to make it recognize general language for user-specified input without providing a complete dictionary along with the input file.
129131		
129132		
129133		
129134		
129134	<i>su</i>	This utility is not useful from shell scripts or typical application programs. (There was also sentiment to avoid security-related utilities.)
129135		
129136	<i>sum</i>	This utility was renamed <i>cksum</i> .
129137	<i>tar</i>	This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.
129138	<i>unpack</i>	This compression program was considered inferior to <i>uncompress</i> .
129139	<i>wall</i>	This utility is terminal-oriented and is not useful in shell scripts or typical applications. It is generally used only by system administrators.
129140		

129141

 *Rationale (Informative)*

129142

Part D:

129143

Portability Considerations

129144

The Open Group

129145

The Institute of Electrical and Electronics Engineers, Inc.

129146

Appendix D

129147

Portability Considerations (Informative)

129148

This section contains information to satisfy various international requirements:

129149

[Section D.1](#) describes perceived user requirements.

129150

[Section D.2](#) (on page 3767) indicates how the facilities of POSIX.1-2017 satisfy those requirements.

129151

129152

[Section D.3](#) (on page 3775) offers guidance to writers of profiles on how the configurable options, limits, and optional behavior of POSIX.1-2017 should be cited in profiles.

129153

129154 **D.1 User Requirements**

129155

This section describes the user requirements that were perceived by the standard developers. The primary source for these requirements was an analysis of historical practice in widespread use, as typified by the base documents for the ISO POSIX-1: 1996 standard.

129156

129157

129158

POSIX.1-2017 addresses the needs of users requiring open systems solutions for source code portability of applications. It currently addresses users requiring open systems solutions for source-code portability of applications involving multi-programming and process management (creating processes, signaling, and so on); access to files and directories in a hierarchy of file systems (opening, reading, writing, deleting files, and so on); access to asynchronous communications ports and other special devices; access to information about other users of the system; facilities supporting applications requiring bounded (realtime) response.

129159

129160

129161

129162

129163

129164

129165

The following users are identified for POSIX.1-2017:

129166

Those employing applications written in high-level languages, such as C, Ada, or FORTRAN.

129167

129168

Users who desire conforming applications that do not necessarily require the characteristics of high-level languages (for example, the speed of execution of compiled languages or the relative security of source code intellectual property inherent in the compilation process).

129169

129170

129171

129172

Users who desire conforming applications that can be developed quickly and can be modified readily without the use of compilers and other system components that may be unavailable on small systems or those without special application development capabilities.

129173

129174

129175

129176

Users who interact with a system to achieve general-purpose time-sharing capabilities common to most business or government offices or academic environments: editing, filing, inter-user communications, printing, and so on.

129177

129178

129179

Users who develop applications for POSIX-conformant systems.

129180

Users who develop applications for UNIX systems.

129181

An acknowledged restriction on applicable users is that they are limited to the group of individuals who are familiar with the style of interaction characteristic of historically-derived systems based on one of the UNIX operating systems (as opposed to other historical systems

129182

129183

129184 with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users
129185 would include program developers, engineers, or general-purpose time-sharing users.

129186 The requirements of users of POSIX.1-2017 can be summarized as a single goal: *application source*
129187 *portability*. The requirements of the user are stated in terms of the requirements of portability of
129188 applications. This in turn becomes a requirement for a standardized set of syntax and semantics
129189 for operations commonly found on many operating systems.

129190 The following sections list the perceived requirements for application portability.

129191 **D.1.1 Configuration Interrogation**

129192 An application must be able to determine whether and how certain optional features are
129193 provided and to identify the system upon which it is running, so that it may appropriately adapt
129194 to its environment.

129195 Applications must have sufficient information to adapt to varying behaviors of the system.

129196 **D.1.2 Process Management**

129197 An application must be able to manage itself, either as a single process or as multiple processes.
129198 Applications must be able to manage other processes when appropriate.

129199 Applications must be able to identify, control, create, and delete processes, and there must be
129200 communication of information between processes and to and from the system.

129201 Applications must be able to use multiple flows of control with a process (threads) and
129202 synchronize operations between these flows of control.

129203 **D.1.3 Access to Data**

129204 Applications must be able to operate on the data stored on the system, access it, and transmit it
129205 to other applications. Information must have protection from unauthorized or accidental access
129206 or modification.

129207 **D.1.4 Access to the Environment**

129208 Applications must be able to access the external environment to communicate their input and
129209 results.

129210 **D.1.5 Access to Determinism and Performance Enhancements**

129211 Applications must have sufficient control of resource allocation to ensure the timeliness of
129212 interactions with external objects.

129213 D.1.6 Operating System-Dependent Profile

129214 The capabilities of the operating system may make certain optional characteristics of the base
129215 language in effect no longer optional, and this should be specified.

129216 D.1.7 I/O Interaction

129217 The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of
129218 POSIX.1-2017 must be specified.

129219 D.1.8 Internationalization Interaction

129220 The effects of the environment of POSIX.1-2017 on the internationalization facilities of the C
129221 language must be specified.

129222 D.1.9 C-Language Extensions

129223 Certain functions in the C language must be extended to support the additional capabilities
129224 provided by POSIX.1-2017.

129225 D.1.10 Command Language

129226 Users should be able to define procedures that combine simple tools and/or applications into
129227 higher-level components that perform to the specific needs of the user. The user should be able
129228 to store, recall, use, and modify these procedures. These procedures should employ a powerful
129229 command language that is used for recurring tasks in conforming applications (scripts) in the
129230 same way that it is used interactively to accomplish one-time tasks. The language and the
129231 utilities that it uses must be consistent between systems to reduce errors and retraining.

129232 D.1.11 Interactive Facilities

129233 Use the system to accomplish individual tasks at an interactive terminal. The interface should be
129234 consistent, intuitive, and offer usability enhancements to increase the productivity of terminal
129235 users, reduce errors, and minimize retraining costs. Online documentation or usage assistance
129236 should be available.

129237 D.1.12 Accomplish Multiple Tasks Simultaneously

129238 Access applications and interactive facilities from a single terminal without requiring serial
129239 execution: switch between multiple interactive tasks; schedule one-time or periodic background
129240 work; display the status of all work in progress or scheduled; influence the priority scheduling
129241 of work, when authorized.

129242 D.1.13 Complex Data Manipulation

129243 Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern
129244 match, select subsets (strings, columns, fields, rows, and so on). These facilities should be
129245 available to both conforming applications and interactive users.

129246 D.1.14 File Hierarchy Manipulation

129247 Create, delete, move/rename, copy, backup/archive, and display files and directories. These
129248 facilities should be available to both conforming applications and interactive users.

129249 D.1.15 Locale Configuration

129250 Customize applications and interactive sessions for the cultural and language conventions of the
129251 user. Employ a wide variety of standard character encodings. These facilities should be available
129252 to both conforming applications and interactive users.

129253 D.1.16 Inter-User Communication

129254 Send messages or transfer files to other users on the same system or other systems on a network.
129255 These facilities should be available to both conforming applications and interactive users.

129256 D.1.17 System Environment

129257 Display information about the status of the system (activities of users and their interactive and
129258 background work, file system utilization, system time, configuration, and presence of optional
129259 facilities) and the environment of the user (terminal characteristics, and so on). Inform the
129260 system operator/administrator of problems. Control access to user files and other resources.

129261 D.1.18 Printing

129262 Output files on a variety of output device classes, accessing devices on local or network-
129263 connected systems. Control (or influence) the formatting, priority scheduling, and output
129264 distribution of work. These facilities should be available to both conforming applications and
129265 interactive users.

129266 D.1.19 Software Development

129267 Develop (create and manage source files, compile/interpret, debug) portable open systems
129268 applications and package them for distribution to, and updating of, other systems.

129269 **D.2 Portability Capabilities**

129270 This section describes the significant portability capabilities of POSIX.1-2017 and indicates how
 129271 the user requirements listed in [Section D.1](#) (on page 3763) are addressed. The capabilities are
 129272 listed in the same format as the preceding user requirements; they are summarized below:

- 129273 Configuration Interrogation
- 129274 Process Management
- 129275 Access to Data
- 129276 Access to the Environment
- 129277 Access to Determinism and Performance Enhancements
- 129278 Operating System-Dependent Profile
- 129279 I/O Interaction
- 129280 Internationalization Interaction
- 129281 C-Language Extensions
- 129282 Command Language
- 129283 Interactive Facilities
- 129284 Accomplish Multiple Tasks Simultaneously
- 129285 Complex Data Manipulation
- 129286 File Hierarchy Manipulation
- 129287 Locale Configuration
- 129288 Inter-User Communication
- 129289 System Environment
- 129290 Printing
- 129291 Software Development

129292 **D.2.1 Configuration Interrogation**

129293 The *uname()* operation provides basic identification of the system. The *sysconf()*, *pathconf()*, and
 129294 *fpathconf()* functions and the *getconf* utility provide means to interrogate the implementation to
 129295 determine how to adapt to the environment in which it is running. These values can be either
 129296 static (indicating that all instances of the implementation have the same value) or dynamic
 129297 (indicating that different instances of the implementation have the different values, or that the
 129298 value may vary for other reasons, such as reconfiguration).

129299 **Unsatisfied Requirements**

129300 None directly. However, as new areas are added, there will be a need for additional capability in
 129301 this area.

129302 D.2.2 Process Management

129303 The *fork()*, *exec* family, *posix_spawn()*, and *posix_spawnp()* functions provide for the creation of
 129304 new processes or the insertion of new applications into existing processes. The *_Exit()*, *_exit()*,
 129305 *exit()*, and *abort()* functions allow for the termination of a process by itself. The *wait()*, *waitid()*,
 129306 and *waitpid()* functions allow one process to deal with the termination of another.

129307 The *times()* function allows for basic measurement of times used by a process. Various
 129308 functions, including *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getgrgid()*, *getgrnam()*, *getlogin()*,
 129309 *getpid()*, *getppid()*, *getpwnam()*, *getpwuid()*, *getuid()*, *lstat()*, and *stat()*, provide for access to the
 129310 identifiers of processes and the identifiers and names of owners of processes (and files).

129311 The various functions operating on environment variables provide for communication of
 129312 information (primarily user-configurable defaults) from a parent to child processes.

129313 The operations on the current working directory control and interrogate the directory from
 129314 which relative pathname searches start. The *umask()* function controls the default protections
 129315 applied to files created by the process.

129316 The *alarm()*, *pause()*, *sleep()*, *ualarm()*, and *usleep()* operations allow the process to suspend until
 129317 a timer has expired or to be notified when a period of time has elapsed. The *time()* operation
 129318 interrogates the current time and date.

129319 The signal mechanism provides for communication of events either from other processes or
 129320 from the environment to the application, and the means for the application to control the effect
 129321 of these events. The mechanism provides for external termination of a process and for a process
 129322 to suspend until an event occurs. The mechanism also provides for a value to be associated with
 129323 an event.

129324 Job control provides a means to group processes and control them as groups, and to control their
 129325 access to the function between the user and the system (the “controlling terminal”). It also
 129326 provides the means to suspend and resume processes.

129327 The Process Scheduling option provides control of the scheduling and priority of a process.

129328 The Message Passing option provides a means for interprocess communication involving small
 129329 amounts of data.

129330 The Memory Management facilities provide control of memory resources and for the sharing of
 129331 memory. This functionality is mandatory on POSIX-conforming systems.

129332 The Threads facilities provide multiple flows of control with a process (threads),
 129333 synchronization between threads (including mutexes, barriers, and spin locks), association of
 129334 data with threads, and controlled cancellation of threads.

129335 The XSI interprocess communications functionality provide an alternate set of facilities to
 129336 manipulate semaphores, message queues, and shared memory. These are provided on XSI-
 129337 conformant systems to support conforming applications developed to run on UNIX systems.

129338 D.2.3 Access to Data

129339 The *open()*, *close()*, *fclose()*, *fopen()*, and *pipe()* functions provide for access to files and data.
 129340 Such files may be regular files, interprocess data channels (pipes), or devices. Additional types
 129341 of objects in the file system are permitted and are being contemplated for standardization.

129342 The *access()*, *chmod()*, *chown()*, *dup()*, *dup2()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lstat()*,
 129343 *readlink()*, *realpath()*, *stat()*, and *utime()* functions allow for control and interrogation of file and
 129344 file-related objects (including symbolic links), and their ownership, protections, and timestamps.

129345 The *fgetc()*, *fputc()*, *fread()*, *fseek()*, *fsetpos()*, *fwrite()*, *getc()*, *getchar()*, *lseek()*, *putchar()*, *putc()*,
 129346 *read()*, and *write()* functions provide for data transfer from the application to files (in all their
 129347 forms).

129348 The *closedir()*, *link()*, *mkdir()*, *opendir()*, *readdir()*, *rename()*, *rmdir()*, *rewinddir()*, and *unlink()*
 129349 functions provide for a complete set of operations on directories. Directories can arbitrarily
 129350 contain other directories, and a single file can be mentioned in more than one directory.

129351 The *faccessat()*, *openat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*, *renameat()*, *readlinkat()*,
 129352 *symlinkat()*, and *unlinkat()* functions allow for race-free and thread-safe file access. The
 129353 motivation for the introduction of these functions was as follows:

129354 Interfaces taking a pathname may be limited by the maximum length of a pathname
 129355 (`{PATH_MAX}`). The absolute path of files can far exceed this length. The alternative
 129356 solution of changing the working directory and using relative pathnames is not thread-
 129357 safe.

129358 A second motivation is that files accessed outside the current working directory are subject
 129359 to attacks caused by the race condition created by changing any of the elements of the
 129360 pathnames used.

129361 A third motivation is to allow application code which makes use of a virtual current
 129362 working directory for each individual thread. In the alternative model there is only one
 129363 current working directory for all threads.

129364 The file-locking mechanism provides for advisory locking (protection during transactions) of
 129365 ranges of bytes (in effect, records) in a file.

129366 The *confstr()*, *fpathconf()*, *pathconf()*, and *sysconf()* functions provide for enquiry as to the
 129367 behavior of the system where variability is permitted.

129368 The asynchronous input and output functions *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*,
 129369 *aio_return()*, *aio_suspend()*, *aio_write()*, and *lio_listio()* provide for initiation and control of
 129370 asynchronous data transfers.

129371 The Synchronized Input and Output option provides for assured commitment of data to media.

129372 D.2.4 Access to the Environment

129373 The operations and types in XBD are provided for access to asynchronous serial devices. The
 129374 primary intended use for these is the controlling terminal for the application (the interaction
 129375 point between the user and the system). They are general enough to be used to control any
 129376 asynchronous serial device. The functions are also general enough to be used with many other
 129377 device types as a user interface when some emulation is provided.

129378 Less detailed access is provided for other device types, but in many instances an application
 129379 need not know whether an object in the file system is a device or a regular file to operate
 129380 correctly.

129381 **Unsatisfied Requirements**

129382 Detailed control of common device classes, specifically magnetic tape, is not provided.

129383 **D.2.5 Bounded (Realtime) Response**129384 The realtime signal functions *sigqueue()*, *sigtimedwait()*, and *sigwaitinfo()* provide queued signals
129385 and the prioritization of the handling of signals.129386 The SCHED_FIFO, SCHED_SPORADIC, and SCHED_RR scheduling policies provide control
129387 over processor allocation.129388 The semaphore functions *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*,
129389 *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*, and *sem_wait()* provide high-
129390 performance synchronization.129391 The memory management functions provide memory locking for control of memory allocation,
129392 file mapping for high performance, and shared memory for high-performance interprocess
129393 communication. The Message Passing option provides for interprocess communication without
129394 being dependent on shared memory.129395 The timers functions *clock_getres()*, *clock_gettime()*, *clock_settime()*, *nanosleep()*, *timer_create()*,
129396 *timer_delete()*, *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* provide functionality to
129397 manipulate clocks and timers and include a high resolution function called *nanosleep()* with a
129398 finer resolution than the *sleep()* function.129399 The timeout functions \nexists *pthread_mutex_timedlock()*, *pthread_rwlock_timedrdlock()*,
129400 *pthread_rwlock_timedwrlock()*, and *sem_timedwait()* \nexists the Typed Memory Objects option and the
129401 Monotonic Clock option provide further facilities for applications to use to obtain predictable
129402 bounded response.129403 **D.2.6 Operating System-Dependent Profile**129404 POSIX.1-2017 makes no distinction between text and binary files. The values of EXIT_SUCCESS
129405 and EXIT_FAILURE are further defined.129406 **Unsatisfied Requirements**129407 None known, but the ISO C standard may contain some additional options that could be
129408 specified.129409 **D.2.7 I/O Interaction**129410 POSIX.1-2017 defines how each of the ISO C standard *stdio* functions interact with the POSIX.1
129411 operations, typically specifying the behavior in terms of POSIX.1 operations.

129412 **Unsatisfied Requirements**

129413 None.

129414 **D.2.8 Internationalization Interaction**

129415 The POSIX.1-2017 environment operations provide a means to define the environment for
 129416 *setlocale()* and time functions such as *ctime()*. The *tzset()* function is provided to set time
 129417 conversion information.

129418 The *nl_langinfo()* function is provided to query locale-specific cultural settings.

129419 The multiple concurrent locale functions *duplocale()*, *freelocale()*, *is*_l()*, *newlocale()*,
 129420 *strcasemp_l()*, *strcoll_l()*, *strfmon_l()*, *strncasemp_l()*, *strxfrm_l()*, *tolower_l()*, *toupper_l()*,
 129421 *towctrans_l()*, *towlower_l()*, *towupper_l()*, *uselocale()*, *wscasemp_l()*, *wscoll_l()*, *wscncasemp_l()*,
 129422 *wcsxfrm_l()*, *wctrans_l()*, and *wctype_l()* are provide to support per-thread locale information.

129423 **Unsatisfied Requirements**

129424 None.

129425 **D.2.9 C-Language Extensions**

129426 The *setjmp()* and *longjmp()* functions are not defined to be cognizant of the signal masks defined
 129427 for POSIX.1. The *sigsetjmp()* and *siglongjmp()* functions are provided to fill this gap.

129428 The *_setjmp()* and *_longjmp()* functions are provided as XSI options to support historic practice.

129429 **Unsatisfied Requirements**

129430 None.

129431 **D.2.10 Command Language**

129432 The shell command language, as described in XCU [Chapter 2](#) (on page 2345), is a common
 129433 language useful in batch scripts, through an API to high-level languages (for the C-Language
 129434 Binding option, *system()* and *popen()*) and through an interactive terminal (see the *sh* utility).
 129435 The shell language has many of the characteristics of a high-level language, but it has been
 129436 designed to be more suitable for user terminal entry and includes interactive debugging
 129437 facilities. Through the use of pipelining, many complex commands can be constructed from
 129438 combinations of data filters and other common components. Shell scripts can be created, stored,
 129439 recalled, and modified by the user with simple editors.

129440 In addition to the basic shell language, the following utilities offer features that simplify and
 129441 enhance programmatic access to the utilities and provide features normally found only in high-
 129442 level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*,
 129443 *time**,⁹ *true*, *wait*, *xargs*, and all of the special built-in utilities in XCU [Section 2.14](#) (on page 2384).

129444 9. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability Utilities
 129445 option. There may be further restrictions on the utilities offered with various configuration option combinations; see the individual utility
 129446 descriptions.

129447 **Unsatisfied Requirements**

129448 None.

129449 **D.2.11 Interactive Facilities**

129450 The utilities offer a common style of command-line interface through conformance to the Utility
 129451 Syntax Guidelines (see XBD [Section 12.2](#), on page 216) and the common utility defaults (see XCU
 129452 [Section 1.4](#), on page 2336). The *sh* utility offers an interactive command-line history and editing
 129453 facility.

129454 The following utilities can be used interactively as well as by scripts; *alias*, *fc*, *mailx*, *unalias*, and
 129455 *write*.

129456 The following utilities in the User Portability Utilities option provide for interactive use: *ex*, *more*,
 129457 and *vi*; the *man* utility offers online access to system documentation.

129458 **Unsatisfied Requirements**

129459 The command line interface to individual utilities is as intuitive and consistent as historical
 129460 practice allows. Work underway based on graphical user interfaces may be more suitable for
 129461 novice or occasional users of the system.

129462 **D.2.12 Accomplish Multiple Tasks Simultaneously**

129463 The shell command language offers background processing through the asynchronous list
 129464 command form; see XCU [Section 2.9](#) (on page 2365).

129465 The *nohup* utility makes background processing more robust and usable.

129466 The *kill* utility can terminate background jobs.

129467 The following utilities support periodic job scheduling, control, and display: *at*, *batch*, *crontab*,
 129468 *nice*, *ps*, and *renice*.

129469 When the User Portability Utilities option is supported, the following utilities allow
 129470 manipulation of jobs: *bg*, *fg*, and *jobs*.

129471 **Unsatisfied Requirements**

129472 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses
 129473 than the job control approach in POSIX.1-2017. See the comments on graphical user interfaces in
 129474 [Section D.2.11](#). The *nice* and *renice* utilities do not necessarily take advantage of complex system
 129475 scheduling algorithms that are supported by the realtime options within POSIX.1-2017.

129476 **D.2.13 Complex Data Manipulation**

129477 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit*, *cut*,
 129478 *dd*, *diff*, *ed*, *ex**, *expand*, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split*, *tabs*, *tail*, *tr*,
 129479 *unexpand*, *uniq*, *uudecode*, *uuencode*, and *wc*.

129480 **Unsatisfied Requirements**

129481 Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in
 129482 the area of SGML may satisfy this.

129483 **D.2.14 File Hierarchy Manipulation**

129484 The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*,
 129485 *cksum*, *cp*, *dd*, *df*, *diff*, *dirname*, *du*, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch*, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*,
 129486 *test*, and *touch*.

129487 **Unsatisfied Requirements**

129488 Some graphical user interfaces offer more intuitive file manager components that allow file
 129489 manipulation through the use of icons for novice users.

129490 **D.2.15 Locale Configuration**

129491 The standard utilities are affected by the various *LC_* variables to achieve locale-dependent
 129492 operation: character classification, collation sequences, regular expressions and shell pattern
 129493 matching, date and time formats, numeric formatting, and monetary formatting. When the
 129494 POSIX2_LOCALEDEF option is supported, applications can provide their own locale definition
 129495 files.

129496 The following utilities address user requirements in this area: *date*, *ed*, *ex**, *find*, *grep*, *locale*,
 129497 *localedef*, *more**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi**.

129498 The *iconv()*, *iconv_close()*, and *iconv_open()* functions are available to allow an application to
 129499 convert character data between supported character sets.

129500 The *genccat* utility and the *catopen()*, *catclose()*, and *catgets()* functions provide for message
 129501 catalog manipulation.

129502 **Unsatisfied Requirements**

129503 Some aspects of multi-byte character and state-encoded character encodings have not yet been
 129504 addressed. The C-language functions, such as *getopt()*, are generally limited to single-byte
 129505 characters. The effect of the *LC_MESSAGES* variable on message formats is only suggested at
 129506 this time.

129507 **D.2.16 Inter-User Communication**

129508 The following utilities address user requirements in this area: *cksum*, *mailx*, *mesg*, *patch*, *pax*, *talk*,
 129509 *uudecode*, *uuencode*, *who*, and *write*.

129510 The historical UUCP utilities are included as a separate UUCP Utilities option.

129511 **Unsatisfied Requirements**

129512 None.

129513 **D.2.17 System Environment**129514 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df*, *du*, *env*,
129515 *getconf*, *id*, *logger*, *logname*, *msg*, *newgrp*, *ps*, *stty*, *tput*, *tty*, *umask*, *uname*, and *who*.129516 The *closelog()*, *openlog()*, *setlogmask()*, and *syslog()* functions provide system logging facilities on
129517 XSI-conformant systems; these are analogous to the *logger* utility.129518 **Unsatisfied Requirements**

129519 None.

129520 **D.2.18 Printing**129521 The following utilities address user requirements in this area: *pr* and *lp*.129522 **Unsatisfied Requirements**

129523 There are no features to control the formatting or scheduling of the print jobs.

129524 **D.2.19 Software Development**129525 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c99*, *ctags*, *fort77*,
129526 *getconf*, *getopts*, *lex*, *localedef*, *make*, *nm*, *od*, *patch*, *pax*, *strings*, *strip*, *time*, and *yacc*.129527 The *system()*, *popen()*, *pclose()*, *regcomp()*, *regex()*, *regerror()*, *regfree()*, *fnmatch()*, *getopt()*,
129528 *glob()*, *globfree()*, *wordexp()*, and *wordfree()* functions allow C-language programmers to access
129529 some of the interfaces used by the utilities, such as argument processing, regular expressions,
129530 and pattern matching.129531 The SCCS source-code control system utilities are available on systems supporting the XSI
129532 Development option.129533 **Unsatisfied Requirements**129534 There are no language-specific development tools related to languages other than C and
129535 FORTRAN. The C tools are more complete and varied than the FORTRAN tools. There is no
129536 data dictionary or other CASE-like development tools.129537 **D.2.20 Future Growth**129538 It is arguable whether or not all functionality to support applications is potentially within the
129539 scope of POSIX.1-2017. As a simple matter of practicality, it cannot be. Areas such as graphics,
129540 application domain-specific functionality, windowing, and so on, should be in unique standards.
129541 As such, they are properly “Unsatisfied Requirements” in terms of providing fully conforming
129542 applications, but ones which are outside the scope of POSIX.1-2017.129543 However, as the standards evolve, certain functionality once considered “exotic” enough to be
129544 part of a separate standard become common enough to be included in a core standard such as
129545 this. Realtime and networking, for example, have both moved from separate standards (with

129546 much difficult cross-referencing) into this standard over time, and although no specific areas
129547 have been identified for inclusion in a future version, such inclusions seem likely.

129548 **D.3 Profiling Considerations**

129549 This section offers guidance to writers of profiles on how the configurable options, limits, and
129550 optional behavior of POSIX.1-2017 should be cited in profiles. Profile writers should consult the
129551 general guidance in POSIX.0 when writing POSIX Standardized Profiles.

129552 The information in this section is an inclusive list of features that should be considered by profile
129553 writers. Subsetting of POSIX.1-2017 should follow XBD [Section 2.1.5.1](#) (on page 21). A set of
129554 profiling options is described in [Appendix E](#) (on page 3789).

129555 **D.3.1 Configuration Options**

129556 There are two set of options suggested by POSIX.1-2017: those for POSIX-conforming systems
129557 and those for X/Open System Interface (XSI) conformance. The requirements for XSI
129558 conformance are documented in the Base Definitions volume of POSIX.1-2017 and not discussed
129559 further here, as they superset the POSIX conformance requirements.

129560 **D.3.2 Configuration Options (Shell and Utilities)**

129561 There are three broad optional configurations for the Shell and Utilities volume of POSIX.1-2017:
129562 basic execution system, development system, and user portability interactive system. The
129563 options to support these, and other minor configuration options, are listed in XBD [Chapter 2](#) (on
129564 page 15). Profile writers should consult the following list and the comments concerning user
129565 requirements addressed by various components in [Section D.2](#) (on page 3767).

129566 POSIX2_UPE

129567 The system supports the User Portability Utilities option.

129568 This option is a requirement for a user portability interactive system. It is required
129569 frequently except for those systems, such as embedded realtime or dedicated application
129570 systems, that support little or no interactive time-sharing work by users or operators. XSI-
129571 conformant systems support this option.

129572 POSIX2_SW_DEV

129573 The system supports the Software Development Utilities option.

129574 This option is required by many systems, even those in which actual software development
129575 does not occur. The *make* utility, in particular, is required by many application software
129576 packages as they are installed onto the system. If POSIX2_C_DEV is supported,
129577 POSIX2_SW_DEV is almost a mandatory requirement because of *ar* and *make*.

129578 POSIX2_C_BIND

129579 The system supports the C-Language Bindings option.

129580 This option is required on some implementations developing complex C applications or on
129581 any system installing C applications in source form that require the functions in this option.
129582 The *system()* and *popen()* functions, in particular, are widely used by applications; the
129583 others are rather more specialized.

- 129584 POSIX2_C_DEV
 129585 The system supports the C-Language Development Utilities option.
- 129586 This option is required by many systems, even those in which actual C-language software
 129587 development does not occur. The *c99* utility, in particular, is required by many application
 129588 software packages as they are installed onto the system. The *lex* and *yacc* utilities are used
 129589 less frequently.
- 129590 POSIX2_FORT_DEV
 129591 The system supports the FORTRAN Development Utilities option
- 129592 As with C, this option is needed on any system developing or installing FORTRAN
 129593 applications in source form.
- 129594 POSIX2_FORT_RUN
 129595 The system supports the FORTRAN Runtime Utilities option.
- 129596 This option is required for some FORTRAN applications that need the *asa* utility to convert
 129597 Hollerith printing statement output. It is unknown how frequently this occurs.
- 129598 POSIX2_LOCALEDEF
 129599 The system supports the creation of locales.
- 129600 This option is needed if applications require their own customized locale definitions to
 129601 operate. It is presently unknown whether many applications are dependent on this.
 129602 However, the option is virtually mandatory for systems in which internationalized
 129603 applications are developed.
- 129604 XSI-conformant systems support this option.
- 129605 POSIX2_PBS
 129606 The system supports the Batch Environment Services and Utilities option.
- 129607 POSIX2_PBS_ACCOUNTING
 129608 The system supports the optional feature of accounting within the Batch Environment
 129609 Services and Utilities option. It will be required in servers that implement the optional
 129610 feature of accounting.
- 129611 POSIX2_PBS_CHECKPOINT
 129612 The system supports the optional feature of checkpoint/restart within the Batch
 129613 Environment Services and Utilities option.
- 129614 POSIX2_PBS_LOCATE
 129615 The system supports the optional feature of locating batch jobs within the Batch
 129616 Environment Services and Utilities option.
- 129617 POSIX2_PBS_MESSAGE
 129618 The system supports the optional feature of sending messages to batch jobs within the Batch
 129619 Environment Services and Utilities option.
- 129620 POSIX2_PBS_TRACK
 129621 The system supports the optional feature of tracking batch jobs within the Batch
 129622 Environment Services and Utilities option.
- 129623 POSIX2_CHAR_TERM
 129624 The system supports at least one terminal type capable of all operations described in
 129625 POSIX.1-2017.
- 129626 On systems with POSIX2_UPE, this option is almost always required. It was developed
 129627 solely to allow certain specialized vendors and user applications to bypass the requirement
 129628 for general-purpose asynchronous terminal support. For example, an application and

129629 system that was suitable for block-mode terminals, such as IBM 3270s, would not need this
 129630 option.
 129631 XSI-conformant systems support this option.

129632 D.3.3 Configurable Limits

129633 Very few of the limits need to be increased for profiles. No profile can cite lower values.
 129634 {POSIX2_BC_BASE_MAX}
 129635 {POSIX2_BC_DIM_MAX}
 129636 {POSIX2_BC_SCALE_MAX}
 129637 {POSIX2_BC_STRING_MAX}
 129638 No increase is anticipated for any of these *bc* values, except for very specialized applications
 129639 involving huge numbers.
 129640 {POSIX2_COLL_WEIGHTS_MAX}
 129641 Some natural languages with complex collation requirements require an increase from the
 129642 default 2 to 4; no higher numbers are anticipated.
 129643 {POSIX2_EXPR_NEST_MAX}
 129644 No increase is anticipated.
 129645 {POSIX2_LINE_MAX}
 129646 This number is much larger than most historical applications have been able to use. At some
 129647 future time, applications may be rewritten to take advantage of even larger values.
 129648 {POSIX2_RE_DUP_MAX}
 129649 No increase is anticipated.
 129650 {POSIX2_VERSION}
 129651 This is actually not a limit, but a standard version stamp. Generally, a profile should specify
 129652 XCU [Chapter 2](#) (on page 2345) by name in the normative references section, not this value.

129653 D.3.4 Configuration Options (System Interfaces)

129654 {NGROUPS_MAX}
 129655 A non-zero value indicates that the implementation supports supplementary groups.
 129656 This option is needed where there is a large amount of shared use of files, but where a
 129657 certain amount of protection is needed. Many profiles¹⁰ are known to require this option; it
 129658 should only be required if needed, but it should never be prohibited.
 129659 _POSIX_ADVISORY_INFO
 129660 The system provides advisory information for file management.
 129661 This option allows the application to specify advisory information that can be used to
 129662 achieve better or even deterministic response time in file manager or input and output
 129663 operations.
 129664 _POSIX_ASYNCHRONOUS_IO
 129665 Support for asynchronous input and output is mandatory in POSIX.1-2017.

129666 10. There are no formally approved profiles of POSIX.1-2017 at the time of publication; the reference here is to various profiles generated by
 129667 private bodies or governments.

- 129668 _POSIX_BARRIERS
 129669 Support for barrier synchronization is mandatory in POSIX.1-2017.
- 129670 This facility allows efficient synchronization of multiple parallel threads in multi-processor
 129671 systems in which the operation is supported in part by the hardware architecture.
- 129672 _POSIX_CHOWN_RESTRICTED
 129673 The system restricts the right to “give away” files to other users. It is mandatory that an
 129674 implementation be able to support this facility in POSIX.1-2017; however, it is recognized
 129675 that implementations need not enable the functionality by default.
- 129676 Some applications expect that they can change the ownership of files in this way. It is
 129677 provided where either security or system account requirements cause this ability to be a
 129678 problem. It is also known to be specified in many profiles.
- 129679 _POSIX_CLOCK_SELECTION
 129680 Support for clock selection is mandatory in POSIX.1-2017.
- 129681 This facility allows applications to request a high resolution sleep in order to suspend a
 129682 thread during a relative time interval, or until an absolute time value, using the desired
 129683 clock. It also allows the application to select the clock used in a *pthread_cond_timedwait()*
 129684 function call.
- 129685 _POSIX_CPUTIME
 129686 The system supports the Process CPU-Time Clocks option.
- 129687 This option allows applications to use a new clock that measures the execution times of
 129688 processes or threads, and the possibility to create timers based upon these clocks, for
 129689 runtime detection (and treatment) of execution time overruns.
- 129690 _POSIX_FSYNC
 129691 The system supports file synchronization requests.
- 129692 This option was created to support historical systems that did not provide the feature.
 129693 Applications that are expecting guaranteed completion of their input and output operations
 129694 should require the *_POSIX_SYNC_IO* option. This option should never be prohibited.
- 129695 XSI-conformant systems support this option.
- 129696 _POSIX_IPV6
 129697 The system supports facilities related to Internet Protocol Version 6 (IPv6).
- 129698 This option was created to allow systems to transition to IPv6.
- 129699 _POSIX_JOB_CONTROL
 129700 Support for job control is mandatory in POSIX.1-2017.
- 129701 Most applications that use it can run when it is not present, although with a degraded level
 129702 of user convenience.
- 129703 _POSIX_MAPPED_FILES
 129704 Support for memory mapped files is mandatory in POSIX.1-2017.
- 129705 This facility provides for the mapping of regular files into the process address space.
- 129706 Both this facility and the Shared Memory Objects option provide shared access to memory
 129707 objects in the process address space. The *mmap()* and *munmap()* functions provide the
 129708 functionality of existing practice for mapping regular files. This functionality was deemed
 129709 unnecessary, if not inappropriate, for embedded systems applications and is expected to be
 129710 optional in subprofiles.

- 129711 `_POSIX_MEMLOCK`
 129712 The system supports the locking of the address space.
- 129713 This option was created to support historical systems that did not provide the feature. It
 129714 should only be required if needed, but it should never be prohibited.
- 129715 `_POSIX_MEMLOCK_RANGE`
 129716 The system supports the locking of specific ranges of the address space.
- 129717 For applications that have well-defined sections that need to be locked and others that do
 129718 not, POSIX.1-2017 supports an optional set of functions to lock or unlock a range of process
 129719 addresses. The following are two reasons for having a means to lock down a specific range:
- 129720 1. An asynchronous event handler function that must respond to external events in a
 129721 deterministic manner such that page faults cannot be tolerated
 - 129722 2. An input/output “buffer” area that is the target for direct-to-process I/O, and the
 129723 overhead of implicit locking and unlocking for each I/O call cannot be tolerated
- 129724 It should only be required if needed, but it should never be prohibited.
- 129725 `_POSIX_MEMORY_PROTECTION`
 129726 Support for memory protection is mandatory in POSIX.1-2017.
- 129727 The provision of this facility typically imposes additional hardware requirements.
- 129728 `_POSIX_PRIORITIZED_IO`
 129729 The system provides prioritization for input and output operations.
- 129730 The use of this option may interfere with the ability of the system to optimize input and
 129731 output throughput. It should only be required if needed, but it should never be prohibited.
- 129732 `_POSIX_MESSAGE_PASSING`
 129733 The system supports the passing of messages between processes.
- 129734 This option was created to support historical systems that did not provide the feature. The
 129735 functionality adds a high-performance XSI interprocess communication facility for local
 129736 communication. It should only be required if needed, but it should never be prohibited.
- 129737 `_POSIX_MONOTONIC_CLOCK`
 129738 The system supports the Monotonic Clock option.
- 129739 This option allows realtime applications to rely on a monotonically increasing clock that
 129740 does not jump backwards, and whose value does not change except for the regular ticking
 129741 of the clock.
- 129742 `_POSIX_PRIORITY_SCHEDULING`
 129743 The system provides priority-based process scheduling.
- 129744 Support of this option provides predictable scheduling behavior, allowing applications to
 129745 determine the order in which processes that are ready to run are granted access to a
 129746 processor. It should only be required if needed, but it should never be prohibited.
- 129747 `_POSIX_REALTIME_SIGNALS`
 129748 Support for realtime signals is mandatory in POSIX.1-2017.
- 129749 This facility provides prioritized, queued signals with associated data values.
- 129750 `_POSIX_REGEX`
 129751 Support for regular expression facilities is mandatory in POSIX.1-2017.

- 129752 _posix_saved_ids
129753 Support for this feature is mandatory in POSIX.1-2017.
- 129754 Certain classes of applications rely on it for proper operation, and there is no alternative
129755 short of giving the application root privileges on most implementations that did not provide
129756 _posix_saved_ids.
- 129757 _posix_semaphores
129758 Support for counting semaphores is mandatory in POSIX.1-2017.
- 129759 _posix_shared_memory_objects
129760 The system supports the mapping of shared memory objects into the process address space.
- 129761 Both this option and the Memory Mapped Files option provide shared access to memory
129762 objects in the process address space. The functions defined under this option provide the
129763 functionality of existing practice for shared memory objects. This functionality was deemed
129764 appropriate for embedded systems applications and, hence, is provided under this option.
129765 It should only be required if needed, but it should never be prohibited.
- 129766 _posix_shell
129767 Support for the *sh* utility command line interpreter is mandatory in POSIX.1-2017.
- 129768 _posix_spawn
129769 The system supports the spawn option.
- 129770 This option provides applications with an efficient mechanism to spawn execution of a new
129771 process.
- 129772 _posix_spinlocks
129773 Support for spin locks is mandatory in POSIX.1-2017.
- 129774 This facility provides a simple and efficient synchronization mechanism for threads
129775 executing in multi-processor systems.
- 129776 _posix_sporadic_server
129777 The system supports the sporadic server scheduling policy.
- 129778 This option provides applications with a new scheduling policy for scheduling aperiodic
129779 processes or threads in hard realtime applications.
- 129780 _posix_synchronized_io
129781 The system supports guaranteed file synchronization.
- 129782 This option was created to support historical systems that did not provide the feature.
129783 Applications that are expecting guaranteed completion of their input and output operations
129784 should require this option, rather than the File Synchronization option. It should only be
129785 required if needed, but it should never be prohibited.
- 129786 _posix_threads
129787 Support for multiple threads of control within a single process is mandatory in
129788 POSIX.1-2017.
- 129789 _posix_thread_attr_stackaddr
129790 The system supports specification of the stack address for a created thread.
- 129791 Applications may take advantage of support of this option for performance benefits, but
129792 dependence on this feature should be minimized. This option should never be prohibited.
- 129793 XSI-conformant systems support this option.

- 129794 _posix_thread_attr_stacksize
129795 The system supports specification of the stack size for a created thread.
- 129796 Applications may require this option in order to ensure proper execution, but such usage
129797 limits portability and dependence on this feature should be minimized. It should only be
129798 required if needed, but it should never be prohibited.
- 129799 XSI-conformant systems support this option.
- 129800 _posix_thread_priority_scheduling
129801 The system provides priority-based thread scheduling.
- 129802 Support of this option provides predictable scheduling behavior, allowing applications to
129803 determine the order in which threads that are ready to run are granted access to a processor.
129804 It should only be required if needed, but it should never be prohibited.
- 129805 _posix_thread_prio_inherit
129806 The system provides mutual-exclusion operations with priority inheritance.
- 129807 Support of this option provides predictable scheduling behavior, allowing applications to
129808 determine the order in which threads that are ready to run are granted access to a processor.
129809 It should only be required if needed, but it should never be prohibited.
- 129810 _posix_thread_prio_protect
129811 The system supports a priority ceiling emulation protocol for mutual-exclusion operations.
- 129812 Support of this option provides predictable scheduling behavior, allowing applications to
129813 determine the order in which threads that are ready to run are granted access to a processor.
129814 It should only be required if needed, but it should never be prohibited.
- 129815 _posix_thread_process_shared
129816 The system provides shared access among multiple processes to synchronization objects.
- 129817 This option was created to support historical systems that did not provide the feature. It
129818 should only be required if needed, but it should never be prohibited.
- 129819 XSI-conformant systems support this option.
- 129820 _posix_thread_safe_functions
129821 Support for thread-safe functions is mandatory in POSIX.1-2017.
- 129822 _posix_thread_sporadic_server
129823 The system supports the thread sporadic server scheduling policy.
- 129824 Support for this option provides applications with a new scheduling policy for scheduling
129825 aperiodic threads in hard realtime applications.
- 129826 _posix_timeouts
129827 Support for timeouts for some blocking services is mandatory in POSIX.1-2017.
- 129828 _posix_timers
129829 Support for higher resolution clocks with multiple timers per process is mandatory in
129830 POSIX.1-2017.
- 129831 This facility is appropriate for applications requiring higher resolution timestamps or
129832 needing to control the timing of multiple activities.
- 129833 _posix_trace
129834 The system supports the Trace option.
- 129835 This option was created to allow applications to perform tracing.

- 129836 `_POSIX_TRACE_EVENT_FILTER`
 129837 The system supports the Trace Event Filter option.
 129838 This option is dependent on support of the Trace option.
- 129839 `_POSIX_TRACE_INHERIT`
 129840 The system supports the Trace Inherit option.
 129841 This option is dependent on support of the Trace option.
- 129842 `_POSIX_TRACE_LOG`
 129843 The system supports the Trace Log option.
 129844 This option is dependent on support of the Trace option.
- 129845 `_POSIX_TYPED_MEMORY_OBJECTS`
 129846 The system supports the Typed Memory Objects option.
 129847 This option was created to allow realtime applications to access different kinds of physical
 129848 memory, and allow processes in these applications to share portions of this memory.

129849 **D.3.5 Configurable Limits**

129850 In general, the configurable limits in the `<limits.h>` header defined in the Base Definitions
 129851 volume of POSIX.1-2017 have been set to minimal values; many applications or implementations
 129852 may require larger values. No profile can cite lower values.

129853 `{AIO_LISTIO_MAX}`

129854 The current minimum is likely to be inadequate for most applications. It is expected that
 129855 this value will be increased by profiles requiring support for list input and output
 129856 operations.

129857 `{AIO_MAX}`

129858 The current minimum is likely to be inadequate for most applications. It is expected that
 129859 this value will be increased by profiles requiring support for asynchronous input and
 129860 output operations.

129861 `{AIO_PRIO_DELTA_MAX}`

129862 The functionality associated with this limit is needed only by sophisticated applications. It
 129863 is not expected that this limit would need to be increased under a general-purpose profile.

129864 `{ARG_MAX}`

129865 The current minimum is likely to need to be increased for profiles, particularly as larger
 129866 amounts of information are passed through the environment. Many implementations are
 129867 believed to support larger values.

129868 `{CHILD_MAX}`

129869 The current minimum is suitable only for systems where a single user is not running
 129870 applications in parallel. It is significantly too low for any system also requiring windows,
 129871 and if `_POSIX_JOB_CONTROL` is specified, it should be raised.

129872 `{CLOCKRES_MIN}`

129873 It is expected that profiles will require a finer granularity clock, perhaps as fine as 1 μ s,
 129874 represented by a value of 1 000 for this limit.

129875 `{DELAYTIMER_MAX}`

129876 It is believed that most implementations will provide larger values.

129877	{LINK_MAX}
129878	For most applications and usage, the current minimum is adequate. Many implementations
129879	have a much larger value, but this should not be used as a basis for raising the value unless
129880	the applications to be used require it.
129881	{LOGIN_NAME_MAX}
129882	This is not actually a limit, but an implementation parameter. No profile should impose a
129883	requirement on this value.
129884	{MAX_CANON}
129885	For most purposes, the current minimum is adequate. Unless high-speed burst serial
129886	devices are used, it should be left as is.
129887	{MAX_INPUT}
129888	See {MAX_CANON}.
129889	{MQ_OPEN_MAX}
129890	The current minimum should be adequate for most profiles.
129891	{MQ_PRIO_MAX}
129892	The current minimum corresponds to the required number of process scheduling priorities.
129893	Many realtime practitioners believe that the number of message priority levels ought to be
129894	the same as the number of execution scheduling priorities.
129895	{NAME_MAX}
129896	Many implementations now support larger values, and many applications and users
129897	assume that larger names can be used. Many existing profiles also specify a larger value.
129898	Specifying this value will reduce the number of conforming implementations, although this
129899	might not be a significant consideration over time. Values greater than 255 should not be
129900	required.
129901	{NGROUPS_MAX}
129902	The value selected will typically be 8 or larger.
129903	{OPEN_MAX}
129904	The historically common value for this has been 20. Many implementations support larger
129905	values. If applications that use larger values are anticipated, an appropriate value should be
129906	specified.
129907	{PAGESIZE}
129908	This is not actually a limit, but an implementation parameter. No profile should impose a
129909	requirement on this value.
129910	{PATH_MAX}
129911	Historically, the minimum has been either 1024 or indefinite, depending on the
129912	implementation. Few applications actually require values larger than 256, but some users
129913	may create file hierarchies that must be accessed with longer paths. This value should only
129914	be changed if there is a clear requirement.
129915	{PIPE_BUF}
129916	The current minimum is adequate for most applications. Historically, it has been larger. If
129917	applications that write single transactions larger than this are anticipated, it should be
129918	increased. Applications that write lines of text larger than this probably do not need it
129919	increased, as the text line is delimited by a <newline>.
129920	{POSIX_VERSION}
129921	This is actually not a limit, but a standard version stamp. Generally, a profile should specify
129922	POSIX.1-2017 by a name in the normative references section, not this value.


- 129923 {PTHREAD_DESTRUCTOR_ITERATIONS}
129924 It is unlikely that applications will need larger values to avoid loss of memory resources.
- 129925 {PTHREAD_KEYS_MAX}
129926 The current value should be adequate for most profiles.
- 129927 {PTHREAD_STACK_MIN}
129928 This should not be treated as an actual limit, but as an implementation parameter. No
129929 profile should impose a requirement on this value.
- 129930 {PTHREAD_THREADS_MAX}
129931 It is believed that most implementations will provide larger values.
- 129932 {RTSIG_MAX}
129933 The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a
129934 32-bit field. It is recognized that most existing implementations define many more signals
129935 than are specified in POSIX.1 and, in fact, many implementations have already exceeded 32
129936 signals (including the ``null signal’’). Support of {_POSIX_RTSIG_MAX} additional signals
129937 may push some implementations over the single 32-bit word line, but is unlikely to push
129938 any implementations that are already over that line beyond the 64 signal line.
- 129939 {SEM_NSEMS_MAX}
129940 The current value should be adequate for most profiles.
- 129941 {SEM_VALUE_MAX}
129942 The current value should be adequate for most profiles.
- 129943 {SSIZE_MAX}
129944 This limit reflects fundamental hardware characteristics (the size of an integer), and should
129945 not be specified unless it is clearly required. Extreme care should be taken to assure that
129946 any value that might be specified does not unnecessarily eliminate implementations
129947 because of accidents of hardware design.
- 129948 {STREAM_MAX}
129949 This limit is very closely related to {OPEN_MAX}. It should never be larger than
129950 {OPEN_MAX}, but could reasonably be smaller for application areas where most files are
129951 not accessed through *stdio*. Some implementations may limit {STREAM_MAX} to 20 but
129952 allow {OPEN_MAX} to be considerably larger. Such implementations should be allowed for
129953 if the applications permit.
- 129954 {TIMER_MAX}
129955 The current limit should be adequate for most profiles, but it may need to be larger for
129956 applications with a large number of asynchronous operations.
- 129957 {TTY_NAME_MAX}
129958 This is not actually a limit, but an implementation parameter. No profile should impose a
129959 requirement on this value.
- 129960 {TZNAME_MAX}
129961 The minimum has been historically adequate, but if longer timezone names are anticipated
129962 (particularly such values as UTC-1), this should be increased.

D.3.6 Optional Behavior

129963
129964
129965
129966
129967
129968
129969
129970

In POSIX.1-2017, there are no instances of the terms unspecified, undefined, implementation-defined, or with the verbs “may” or “need not”, that the standard developers anticipate or sanction as suitable for profile or test method citation. All of these are merely warnings to conforming applications to avoid certain areas that can vary from system to system, and even over time on the same system. In many cases, these terms are used explicitly to support extensions, but profiles should not anticipate and require such extensions; future versions of this standard may do so.

129971

 *Rationale (Informative)*

129972

Part E:

129973

Subprofiling Considerations

129974

The Open Group

129975

The Institute of Electrical and Electronics Engineers, Inc.

Subprofiling Considerations (Informative)

129978 This section contains further information to satisfy the requirement that the project scope enable
 129979 subprofiling of POSIX.1-2017. The approach taken is to include a general requirement in
 129980 normative text regarding subprofiling and to include an informative section (here) containing a
 129981 proposed set of subprofiling options.

129982 E.1 Subprofiling Option Groups

129983 The following Option Groups¹¹ are defined to support profiling. Systems claiming support to
 129984 POSIX.1-2017 need not implement these options apart from the requirements stated in XBD
 129985 Section 2.1.3 (on page 17). These Option Groups allow profiles to subset the System Interfaces
 129986 volume of POSIX.1-2017 by collecting sets of related functions.

129987 POSIX_ASYNC_HRONOUS_IO: Asynchronous Input and Output Functions

129988 *aio_cancel()*, *aio_error()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_suspend()*, *aio_write()*,
 129989 *lio_listio()*

129990 POSIX_BARRIERS: Barriers

129991 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*, *pthread_barrierattr()*

129992 POSIX_C_LANG_JUMP: Jump Functions

129993 *longjmp()*, *setjmp()*

129994 POSIX_C_LANG_MATH: Maths Library

129995 *acos()*, *acosf()*, *acosh()*, *acoshf()*, *acoshl()*, *acosl()*, *asin()*, *asinf()*, *asinh()*, *asinhf()*, *asinhl()*,
 129996 *asinl()*, *atan()*, *atan2()*, *atan2f()*, *atan2l()*, *atanf()*, *atanh()*, *atanhf()*, *atanhl()*, *atanl()*, *cabs()*,
 129997 *cabsf()*, *cabsl()*, *cacos()*, *cacosf()*, *cacosh()*, *cacoshf()*, *cacoshl()*, *cacosl()*, *carg()*, *cargf()*, *cargl()*,
 129998 *casin()*, *casinf()*, *casinh()*, *casinhf()*, *casinhl()*, *casinl()*, *catan()*, *catanf()*, *catanh()*, *catanhf()*,
 129999 *catanhl()*, *catanl()*, *cbrt()*, *cbrtf()*, *cbrtl()*, *ccos()*, *ccosf()*, *ccosh()*, *ccoshf()*, *ccoshl()*, *ccosl()*,
 130000 *ceil()*, *ceilf()*, *ceill()*, *cexp()*, *cexpf()*, *cexpl()*, *cimag()*, *cimagf()*, *cimagl()*, *clog()*, *clogf()*, *clogl()*,
 130001 *conj()*, *conjf()*, *conjl()*, *copysign()*, *copysignf()*, *copysignl()*, *cos()*, *cosf()*, *cosh()*, *coshf()*,
 130002 *coshl()*, *cosl()*, *cpow()*, *cpowf()*, *cpowl()*, *cproj()*, *cprojf()*, *cprojl()*, *creal()*, *crealf()*, *creall()*,
 130003 *csin()*, *csinf()*, *csinh()*, *csinhf()*, *csinhl()*, *csinl()*, *csqrt()*, *csqrtf()*, *csqrtl()*, *ctan()*, *ctanf()*,
 130004 *ctanh()*, *ctanhf()*, *ctanhl()*, *ctanl()*, *erf()*, *erfc()*, *erfcf()*, *erfcl()*, *erff()*, *erfl()*, *exp()*, *exp2()*,
 130005 *exp2f()*, *exp2l()*, *expf()*, *expl()*, *expm1()*, *expm1f()*, *expm1l()*, *fabs()*, *fabsf()*, *fabsl()*, *fdim()*,
 130006 *fdimf()*, *fdiml()*, *floor()*, *floorf()*, *floorl()*, *fma()*, *fmaf()*, *fmal()*, *fmax()*, *fmaxf()*, *fmaxl()*, *fmin()*,
 130007 *fminf()*, *fminl()*, *fmod()*, *fmodf()*, *fmodl()*, *fpclassify()*, *frexp()*, *frexpf()*, *frexpl()*, *hypot()*,
 130008 *hypotf()*, *hypotl()*, *ilogb()*, *ilogbf()*, *ilogbl()*, *isfinite()*, *isgreater()*, *isgreaterequal()*, *isinf()*,
 130009 *isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *ldexp()*, *ldexpf()*,
 130010 *ldexpl()*, *lgamma()*, *lgammaf()*, *lgammal()*, *llrint()*, *llrintf()*, *llrintl()*, *llround()*, *llroundf()*,
 130011 *llroundl()*, *log()*, *log10()*, *log10f()*, *log10l()*, *log1p()*, *log1pf()*, *log1pl()*, *log2()*, *log2f()*, *log2l()*,
 130012 *logb()*, *logbf()*, *logbl()*, *logf()*, *logl()*, *lrint()*, *lrintf()*, *lrintl()*, *lround()*, *lroundf()*, *lroundl()*,
 130013 *modf()*, *modff()*, *modfl()*, *nan()*, *nanf()*, *nanl()*, *nearbyint()*, *nearbyintf()*, *nearbyintl()*,
 130014 *nextafter()*, *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, *nexttowardl()*, *pow()*, *powf()*,

130015 11. These are modeled on the Units of Functionality from IEEE Std 1003.13-1998.

130016 *powl()*, *remainder()*, *remainderf()*, *remainderl()*, *remquo()*, *remquof()*, *remquol()*, *rint()*, *rintf()*,
 130017 *rintl()*, *round()*, *roundf()*, *roundl()*, *scalbln()*, *scalblnf()*, *scalblnl()*, *scalbn()*, *scalbnf()*,
 130018 *scalbnl()*, *signbit()*, *sin()*, *sinf()*, *sinh()*, *sinhf()*, *sinhl()*, *sinl()*, *sqrt()*, *sqrtf()*, *sqrtl()*, *tan()*,
 130019 *tanf()*, *tanh()*, *tanhf()*, *tanhL()*, *tanl()*, *tgamma()*, *tgammaf()*, *tgammaL()*, *trunc()*, *truncf()*,
 130020 *truncl()*

130021 POSIX_C_LANG_SUPPORT: General ISO C Library

130022 *abs()*, *asctime()*, *atof()*, *atoi()*, *atol()*, *atoll()*, *bsearch()*, *calloc()*, *ctime()*, *difftime()*, *div()*,
 130023 *feclearexcept()*, *fegetenv()*, *fegetexceptflag()*, *fegetround()*, *fehldexcept()*, *feraiseexcept()*,
 130024 *fesetenv()*, *fesetexceptflag()*, *fesetround()*, *fetestexcept()*, *feupdateenv()*, *free()*, *gmtime()*,
 130025 *imaxabs()*, *imaxdiv()*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*,
 130026 *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *labs()*, *ldiv()*, *llabs()*, *lldiv()*, *localeconv()*,
 130027 *localtime()*, *malloc()*, *memchr()*, *memcmp()*, *memcpy()*, *memmove()*, *memset()*, *mktime()*,
 130028 *qsort()*, *rand()*, *realloc()*, *setlocale()*, *snprintf()*, *sprintf()*, *srand()*, *sscanf()*, *strcat()*, *strchr()*,
 130029 *strcmp()*, *strcoll()*, *strcpy()*, *strcsn()*, *strerror()*, *strftime()*, *strlen()*, *strncat()*, *strncpy()*,
 130030 *strncpy()*, *strpbrk()*, *strrchr()*, *strspn()*, *strstr()*, *strtod()*, *strtof()*, *strtoimax()*, *strtok()*, *strtol()*,
 130031 *strtol()*, *strtoll()*, *strtoul()*, *strtoull()*, *strtoumax()*, *strxfrm()*, *time()*, *tolower()*, *toupper()*,
 130032 *tzname*, *tzset()*, *va_arg()*, *va_copy()*, *va_end()*, *va_start()*, *vsprintf()*, *vsscanf()*

130033 POSIX_C_LANG_SUPPORT_R: Thread-Safe General ISO C Library

130034 *asctime_r()*, *ctime_r()*, *gmtime_r()*, *localtime_r()*, *rand_r()*, *strerror_r()*, *strtok_r()*

130035 POSIX_C_LANG_WIDE_CHAR: Wide-Character ISO C Library

130036 *btowc()*, *iswalnum()*, *iswalphalower()*, *iswblank()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*,
 130037 *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *mblen()*, *mbrlen()*,
 130038 *mbrtowc()*, *mbsinit()*, *mbsrtowcs()*, *mbstowcs()*, *mbtowc()*, *swprintf()*, *swscanf()*, *towctrans()*,
 130039 *towlower()*, *towupper()*, *vwprintf()*, *vwscanf()*, *wcrtomb()*, *wcscat()*, *wcschr()*, *wcscmp()*,
 130040 *wcscoll()*, *wcscpy()*, *wcscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*,
 130041 *wcspbrk()*, *wcsrchr()*, *wcstombs()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstof()*, *wcstoimax()*,
 130042 *wcstok()*, *wcstol()*, *wcstold()*, *wcstoll()*, *wcstombs()*, *wcstoul()*, *wcstoull()*, *wcstoumax()*,
 130043 *wcsxfrm()*, *wctob()*, *wctomb()*, *wctrans()*, *wctype()*, *wmemchr()*, *wmemcmp()*, *wmemcpy()*,
 130044 *wmemmove()*, *wmemset()*

130045 POSIX_C_LANG_WIDE_CHAR_EXT: Extended Wide-Character ISO C Library

130046 *mbsnrtowcs()*, *wcpcpy()*, *wcpncpy()*, *wcscasecmp()*, *wcsdup()*, *wcsncasecmp()*, *wcsnlen()*,
 130047 *wcsnrtombs()*

130048 POSIX_C_LIB_EXT: General C Library Extension

130049 *fnmatch()*, *getopt()*, *getsubopt()*, *optarg*, *opterr*, *optind*, *optopt*, *stpcpy()*, *stpncpy()*, *strcasecmp()*,
 130050 *strdup()*, *strfmon()*, *strncasecmp()*, *strndup()*, *strnlen()*

130051 POSIX_CLOCK_SELECTION: Clock Selection

130052 *clock_nanosleep()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*

130053 POSIX_DEVICE_IO: Device Input and Output

130054 *FD_CLR()*, *FD_ISSET()*, *FD_SET()*, *FD_ZERO()*, *clearerr()*, *close()*, *fclose()*, *fdopen()*, *feof()*,
 130055 *ferror()*, *fflush()*, *fgetc()*, *fgets()*, *fileno()*, *fopen()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *freopen()*,
 130056 *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *open()*, *perror()*, *poll()*, *printf()*, *pread()*, *pselect()*,
 130057 *putc()*, *putchar()*, *puts()*, *pwrite()*, *read()*, *scanf()*, *select()*, *setbuf()*, *setvbuf()*, *stderr*, *stdin*,
 130058 *stdout*, *ungetc()*, *vfprintf()*, *vscanf()*, *vprintf()*, *vscanf()*, *write()*

130059 POSIX_DEVICE_IO_EXT: Extended Device Input and Output

130060 *dprintf()*, *fmemopen()*, *open_memstream()*, *vdprintf()*

130061 POSIX_DEVICE_SPECIFIC: General Terminal

130062 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *ctermid()*, *isatty()*, *tcdrain()*, *tcflow()*,
 130063 *tcflush()*, *tcgetattr()*, *tcsendbreak()*, *tcsetattr()*, *ttyname()*

130064	POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal
130065	<i>ttyname_r()</i>
130066	POSIX_DYNAMIC_LINKING: Dynamic Linking
130067	<i>dlclose(), dlerror(), dlopen(), dlsym()</i>
130068	POSIX_FD_MGMT: File Descriptor Management
130069	<i>dup(), dup2(), fcntl(), fgetpos(), fseek(), fseeko(), fsetpos(), ftell(), ftello(), ftruncate(), lseek(),</i>
130070	<i>rewind()</i>
130071	POSIX_FIFO: FIFO
130072	<i>mkfifo()</i>
130073	POSIX_FIFO_FD: FIFO File Descriptor Routines
130074	<i>mkfifoat(), mknodat()</i>
130075	POSIX_FILE_ATTRIBUTES: File Attributes
130076	<i>chmod(), chown(), fchmod(), fchown(), umask()</i>
130077	POSIX_FILE_ATTRIBUTES_FD: File Attributes File Descriptor Routines
130078	<i>fchmodat(), fchownat()</i>
130079	POSIX_FILE_LOCKING: Thread-Safe Stdio Locking
130080	<i>flockfile(), frylockfile(), funlockfile(), getc_unlocked(), getchar_unlocked(), putc_unlocked(),</i>
130081	<i>putchar_unlocked()</i>
130082	POSIX_FILE_SYSTEM: File System
130083	<i>access(), chdir(), closedir(), creat(), fchdir(), fpathconf(), fstat(), fstatvfs(), getcwd(), link(),</i>
130084	<i>mkdir(), mkstemp(), opendir(), pathconf(), readdir(), remove(), rename(), rewinddir(), rmdir(),</i>
130085	<i>stat(), statvfs(), tmpfile(), tmpnam(), truncate(), unlink(), utime()</i>
130086	POSIX_FILE_SYSTEM_EXT: File System Extensions
130087	<i>alphasort(), dirfd(), getdelim(), getline(), mkdtemp(), scandir()</i>
130088	POSIX_FILE_SYSTEM_FD: File System File Descriptor Routines
130089	<i>faccessat(), fdopendir(), fstatat(), linkat(), mkdirat(), openat(), renameat(), unlinkat(),</i>
130090	<i>utimensat()</i>
130091	POSIX_FILE_SYSTEM_GLOB: File System Glob Expansion
130092	<i>glob(), globfree()</i>
130093	POSIX_FILE_SYSTEM_R: Thread-Safe File System
130094	<i>readdir_r()</i>
130095	POSIX_I18N: Internationalization
130096	<i>catclose(), catgets(), catopen(), iconv(), iconv_close(), iconv_open(), nl_langinfo()</i>
130097	POSIX_JOB_CONTROL: Job Control
130098	<i>setpgid(), tcgetpgrp(), tcsetpgrp(), tcgetsid()</i>
130099	POSIX_MAPPED_FILES: Memory Mapped Files
130100	<i>mmap(), munmap()</i>
130101	POSIX_MEMORY_PROTECTION: Memory Protection
130102	<i>mprotect()</i>
130103	POSIX_MULTI_CONCURRENT_LOCALES: Multiple Concurrent Locales
130104	<i>duplocale(), freelocale(), isalnum_l(), isalpha_l(), isblank_l(), iscntrl_l(), isdigit_l(), isgraph_l(),</i>
130105	<i>islower_l(), isprint_l(), ispunct_l(), isspace_l(), isupper_l(), iswalnum_l(), iswalnum_l(),</i>
130106	<i>iswblank_l(), iswcntrl_l(), iswctype_l(), iswdigit_l(), iswgraph_l(), iswlower_l(), iswprint_l(),</i>
130107	<i>iswpunct_l(), iswspace_l(), iswupper_l(), iswxdigit_l(), isxdigit_l(), newlocale(), strcasecmp_l(),</i>

130108	<i>strcoll_1()</i> , <i>strfmon_1()</i> , <i>strncasecmp_1()</i> , <i>strxfrm_1()</i> , <i>tolower_1()</i> , <i>toupper_1()</i> , <i>towctrans_1()</i> ,
130109	<i>towlower()</i> , <i>towupper()</i> , <i>uselocale()</i> , <i>wcscasecmp_1()</i> , <i>wcscoll_1()</i> , <i>wcsncasecmp_1()</i> , <i>wcsxfrm_1()</i> ,
130110	<i>wctrans_1()</i> , <i>wctype_1()</i>
130111	POSIX_MULTI_PROCESS: Multiple Processes
130112	<i>_Exit()</i> , <i>_exit()</i> , <i>assert()</i> , <i>atexit()</i> , <i>clock()</i> , <i>execl()</i> , <i>execle()</i> , <i>execlp()</i> , <i>execv()</i> , <i>execve()</i> , <i>execvp()</i> ,
130113	<i>exit()</i> , <i>fork()</i> , <i>getpgrp()</i> , <i>getpgid()</i> , <i>getpid()</i> , <i>getppid()</i> , <i>getsid()</i> , <i>setsid()</i> , <i>sleep()</i> , <i>times()</i> , <i>wait()</i> ,
130114	<i>waitid()</i> , <i>waitpid()</i>
130115	POSIX_MULTI_PROCESS_FD: Multiple Processes File Descriptor Routines
130116	<i>fexecve()</i>
130117	POSIX_NETWORKING: Networking
130118	<i>accept()</i> , <i>bind()</i> , <i>connect()</i> , <i>endhostent()</i> , <i>endnetent()</i> , <i>endprotoent()</i> , <i>endservent()</i> ,
130119	<i>freeaddrinfo()</i> , <i>gai_strerror()</i> , <i>getaddrinfo()</i> , <i>gethostent()</i> , <i>gethostname()</i> , <i>getnameinfo()</i> ,
130120	<i>getnetbyaddr()</i> , <i>getnetbyname()</i> , <i>getnetent()</i> , <i>getpeername()</i> , <i>getprotobyname()</i> ,
130121	<i>getprotobynumber()</i> , <i>getprotoent()</i> , <i>getserbyname()</i> , <i>getservbyname()</i> , <i>getservbyport()</i> , <i>getservent()</i> ,
130122	<i>getsockname()</i> , <i>getsockopt()</i> , <i>htonl()</i> , <i>htons()</i> , <i>if_freenameindex()</i> , <i>if_indextoname()</i> ,
130123	<i>if_nameindex()</i> , <i>if_nametoindex()</i> , <i>inet_addr()</i> , <i>inet_ntoa()</i> , <i>inet_ntop()</i> , <i>inet_pton()</i> , <i>listen()</i> ,
130124	<i>ntohl()</i> , <i>ntohs()</i> , <i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , <i>sendto()</i> , <i>sethostent()</i> ,
130125	<i>setnetent()</i> , <i>setprotoent()</i> , <i>setservent()</i> , <i>setsockopt()</i> , <i>shutdown()</i> , <i>socket()</i> , <i>socketatmark()</i> ,
130126	<i>socketpair()</i>
130127	POSIX_PIPE: Pipe
130128	<i>pipe()</i>
130129	POSIX_ROBUST_MUTEXES: Robust Mutexes
130130	<i>pthread_mutex_consistent()</i> , <i>pthread_mutexattr_getrobust()</i> , <i>pthread_mutexattr_setrobust()</i>
130131	POSIX_REALTIME_SIGNALS: Realtime Signals
130132	<i>sigqueue()</i> , <i>sigtimedwait()</i> , <i>sigwaitinfo()</i>
130133	POSIX_REGEX: Regular Expressions
130134	<i>regcomp()</i> , <i>regerror()</i> , <i>regexec()</i> , <i>regfree()</i>
130135	POSIX_RW_LOCKS: Reader Writer Locks
130136	<i>pthread_rwlock_destroy()</i> , <i>pthread_rwlock_init()</i> , <i>pthread_rwlock_rdlock()</i> ,
130137	<i>pthread_rwlock_timedrdlock()</i> , <i>pthread_rwlock_timedwrlock()</i> , <i>pthread_rwlock_tryrdlock()</i> ,
130138	<i>pthread_rwlock_trywrlock()</i> , <i>pthread_rwlock_unlock()</i> , <i>pthread_rwlock_wrlock()</i> ,
130139	<i>pthread_rwlockattr_destroy()</i> , <i>pthread_rwlockattr_init()</i> , <i>pthread_rwlockattr_getpshared()</i> ,
130140	<i>pthread_rwlockattr_setpshared()</i>
130141	POSIX_SEMAPHORES: Semaphores
130142	<i>sem_close()</i> , <i>sem_destroy()</i> , <i>sem_getvalue()</i> , <i>sem_init()</i> , <i>sem_open()</i> , <i>sem_post()</i> ,
130143	<i>sem_timedwait()</i> , <i>sem_trywait()</i> , <i>sem_unlink()</i> , <i>sem_wait()</i>
130144	POSIX_SHELL_FUNC: Shell and Utilities
130145	<i>pclose()</i> , <i>popen()</i> , <i>system()</i> , <i>wordexp()</i> , <i>wordfree()</i>
130146	POSIX_SIGNAL_JUMP: Signal Jump Functions
130147	<i>siglongjmp()</i> , <i>sigsetjmp()</i>
130148	POSIX_SIGNALS: Signals
130149	<i>abort()</i> , <i>alarm()</i> , <i>kill()</i> , <i>pause()</i> , <i>raise()</i> , <i>sigaction()</i> , <i>sigaddset()</i> , <i>sigdelset()</i> , <i>sigemptyset()</i> ,
130150	<i>sigfillset()</i> , <i>sigismember()</i> , <i>signal()</i> , <i>sigpending()</i> , <i>sigprocmask()</i> , <i>sigsuspend()</i> , <i>sigwait()</i>
130151	POSIX_SIGNALS_EXT: Extended Signals
130152	<i>psignal()</i> , <i>psiginfo()</i> , <i>strsignal()</i>

130153	POSIX_SINGLE_PROCESS: Single Process
130154	<i>confstr()</i> , <i>environ</i> , <i>errno</i> , <i>getenv()</i> , <i>setenv()</i> , <i>sysconf()</i> , <i>uname()</i> , <i>unsetenv()</i>
130155	POSIX_SPIN_LOCKS: Spin Locks
130156	<i>pthread_spin_destroy()</i> , <i>pthread_spin_init()</i> , <i>pthread_spin_lock()</i> , <i>pthread_spin_trylock()</i> ,
130157	<i>pthread_spin_unlock()</i>
130158	POSIX_SYMBOLIC_LINKS: Symbolic Links
130159	<i>lchown()</i> , ¹² <i>lstat()</i> , <i>readlink()</i> , <i>symlink()</i>
130160	POSIX_SYMBOLIC_LINKS_FD: Symbolic Links File Descriptor Routines
130161	<i>readlinkat()</i> , <i>symlinkat()</i>
130162	POSIX_SYSTEM_DATABASE: System Database
130163	<i>getgrgid()</i> , <i>getgrnam()</i> , <i>getpwnam()</i> , <i>getpwuid()</i>
130164	POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database
130165	<i>getgrgid_r()</i> , <i>getgrnam_r()</i> , <i>getpwnam_r()</i> , <i>getpwuid_r()</i>
130166	POSIX_THREADS_BASE: Base Threads
130167	<i>pthread_atfork()</i> , <i>pthread_attr_destroy()</i> , <i>pthread_attr_getdetachstate()</i> ,
130168	<i>pthread_attr_getschedparam()</i> , <i>pthread_attr_init()</i> , <i>pthread_attr_setdetachstate()</i> ,
130169	<i>pthread_attr_setschedparam()</i> , <i>pthread_cancel()</i> , <i>pthread_cleanup_pop()</i> , <i>pthread_cleanup_push()</i> ,
130170	<i>pthread_cond_broadcast()</i> , <i>pthread_cond_destroy()</i> , <i>pthread_cond_init()</i> , <i>pthread_cond_signal()</i> ,
130171	<i>pthread_cond_timedwait()</i> , <i>pthread_cond_wait()</i> , <i>pthread_condattr_destroy()</i> ,
130172	<i>pthread_condattr_init()</i> , <i>pthread_create()</i> , <i>pthread_detach()</i> , <i>pthread_equal()</i> , <i>pthread_exit()</i> ,
130173	<i>pthread_getspecific()</i> , <i>pthread_join()</i> , <i>pthread_key_create()</i> , <i>pthread_key_delete()</i> , <i>pthread_kill()</i> ,
130174	<i>pthread_mutex_destroy()</i> , <i>pthread_mutex_init()</i> , <i>pthread_mutex_lock()</i> ,
130175	<i>pthread_mutex_timedlock()</i> , <i>pthread_mutex_trylock()</i> , <i>pthread_mutex_unlock()</i> ,
130176	<i>pthread_mutexattr_destroy()</i> , <i>pthread_mutexattr_init()</i> , <i>pthread_once()</i> , <i>pthread_self()</i> ,
130177	<i>pthread_setcancelstate()</i> , <i>pthread_setcanceltype()</i> , <i>pthread_setspecific()</i> , <i>pthread_sigmask()</i> ,
130178	<i>pthread_testcancel()</i>
130179	POSIX_THREADS_EXT: Extended Threads
130180	<i>pthread_attr_getguardsize()</i> , <i>pthread_attr_setguardsize()</i> , <i>pthread_mutexattr_gettype()</i> ,
130181	<i>pthread_mutexattr_settype()</i>
130182	POSIX_TIMERS: Timers
130183	<i>clock_getres()</i> , <i>clock_gettime()</i> , <i>clock_settime()</i> , <i>nanosleep()</i> , <i>timer_create()</i> , <i>timer_delete()</i> ,
130184	<i>timer_getoverrun()</i> , <i>timer_gettime()</i> , <i>timer_settime()</i>
130185	POSIX_USER_GROUPS: User and Group
130186	<i>getegid()</i> , <i>geteuid()</i> , <i>getgid()</i> , <i>getgroups()</i> , <i>getlogin()</i> , <i>getuid()</i> , <i>setegid()</i> , <i>seteuid()</i> , <i>setgid()</i> ,
130187	<i>setuid()</i>
130188	POSIX_USER_GROUPS_R: Thread-Safe User and Group
130189	<i>getlogin_r()</i>
130190	POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output
130191	<i>fgetwc()</i> , <i>fgetwts()</i> , <i>fputwc()</i> , <i>fputwts()</i> , <i>fwide()</i> , <i>fwprintf()</i> , <i>fwscanf()</i> , <i>getwc()</i> , <i>getwchar()</i> ,
130192	<i>putwc()</i> , <i>putwchar()</i> , <i>ungetwc()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>wprintf()</i> ,
130193	<i>wscanf()</i>
130194	XSI_C_LANG_SUPPORT: XSI General C Library
130195	<i>_tolower()</i> , <i>_toupper()</i> , <i>a64l()</i> , <i>daylight()</i> , <i>drand48()</i> , <i>erand48()</i> , <i>ffs()</i> , <i>getdate()</i> , <i>hcreate()</i> ,
130196	<i>hdestroy()</i> , <i>hsearch()</i> , <i>initstate()</i> , <i>insque()</i> , <i>isascii()</i> , <i>jrand48()</i> , <i>l64a()</i> , <i>lcong48()</i> , <i>lfind()</i> ,
130197	<i>lrand48()</i> , <i>lsearch()</i> , <i>memccpy()</i> , <i>mrand48()</i> , <i>nrand48()</i> , <i>random()</i> , <i>remque()</i> , <i>seed48()</i> ,

130198 12. The *lchown()* function also depends on POSIX_FILE_ATTRIBUTES.

130199	<i>setstate()</i> , <i>signgam</i> , <i>srand48()</i> , <i>srandom()</i> , <i>strptime()</i> , <i>swab()</i> , <i>tdelete()</i> , <i>tfind()</i> , <i>timezone()</i> ,
130200	<i>toascii()</i> , <i>tsearch()</i> , <i>twalk()</i>
130201	XSI_DBM: XSI Database Management
130202	<i>dbm_clearerr()</i> , <i>dbm_close()</i> , <i>dbm_delete()</i> , <i>dbm_error()</i> , <i>dbm_fetch()</i> , <i>dbm_firstkey()</i> ,
130203	<i>dbm_nextkey()</i> , <i>dbm_open()</i> , <i>dbm_store()</i>
130204	XSI_DEVICE_IO: XSI Device Input and Output
130205	<i>fntmsg()</i> , <i>readv()</i> , <i>writew()</i>
130206	XSI_DEVICE_SPECIFIC: XSI General Terminal
130207	<i>grantpt()</i> , <i>posix_openpt()</i> , <i>ptsname()</i> , <i>unlockpt()</i>
130208	XSI_FILE_SYSTEM: XSI File System
130209	<i>basename()</i> , <i>dirname()</i> , <i>ftw()</i> , <i>lockf()</i> , <i>mknod()</i> , <i>nftw()</i> , <i>realpath()</i> , <i>seekdir()</i> , <i>sync()</i> , <i>telldir()</i> ,
130210	<i>tempnam()</i>
130211	XSI_IPC: XSI Interprocess Communication
130212	<i>ftok()</i> , <i>msgctl()</i> , <i>msgget()</i> , <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semctl()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmctl()</i> ,
130213	<i>shmdt()</i> , <i>shmget()</i>
130214	XSI_JUMP: XSI Jump Functions
130215	<i>_longjmp()</i> , <i>_setjmp()</i>
130216	XSI_MATH: XSI Maths Library
130217	<i>j0()</i> , <i>j1()</i> , <i>jn()</i> , <i>y0()</i> , <i>y1()</i> , <i>yn()</i>
130218	XSI_MULTI_PROCESS: XSI Multiple Process
130219	<i>getpriority()</i> , <i>getrlimit()</i> , <i>getrusage()</i> , <i>nice()</i> , <i>setpgrp()</i> , <i>setpriority()</i> , <i>setrlimit()</i> , <i>ulimit()</i>
130220	XSI_SIGNALS: XSI Signal
130221	<i>killpg()</i> , <i>sigaltstack()</i> , <i>sighold()</i> , <i>sigignore()</i> , <i>siginterrupt()</i> , <i>sigpause()</i> , <i>sigrelse()</i> , <i>sigset()</i>
130222	XSI_SINGLE_PROCESS: XSI Single Process
130223	<i>gethostid()</i> , <i>gettimeofday()</i> , <i>putenv()</i>
130224	XSI_SYSTEM_DATABASE: XSI System Database
130225	<i>endpwent()</i> , <i>getpwent()</i> , <i>setpwent()</i>
130226	XSI_SYSTEM_LOGGING: XSI System Logging
130227	<i>closelog()</i> , <i>openlog()</i> , <i>setlogmask()</i> , <i>syslog()</i>
130228	XSI_THREADS_EXT: XSI Threads Extensions
130229	<i>pthread_attr_getstack()</i> , <i>pthread_attr_setstack()</i> , <i>pthread_getconcurrency()</i> ,
130230	<i>pthread_setconcurrency()</i>
130231	XSI_TIMERS: XSI Timers
130232	<i>getitimer()</i> , <i>setitimer()</i>
130233	XSI_USER_GROUPS: XSI User and Group
130234	<i>endgrent()</i> , <i>endutxent()</i> , <i>getgrent()</i> , <i>getutxent()</i> , <i>getutxid()</i> , <i>getutxline()</i> , <i>pututxline()</i> ,
130235	<i>setgrent()</i> , <i>setregid()</i> , <i>setreuid()</i> , <i>setutxent()</i>
130236	XSI_WIDE_CHAR: XSI Wide-Character Library
130237	<i>wcswidth()</i> , <i>wcwidth()</i>

Index

(time) resolution	86
/	197
/dev	197
/dev/console	197
/dev/null	197
/dev/tty	197, 3497
/etc/passwd	3512
/tmp	197
< aio.h>	220
< alert>	34
< apostrophe>	35
< arpa/inet.h>	222
< assert.h>	223
< backspace>	39
< blank>	44
< carriage-return>	46
< circum ex>	48
< complex.h>	224
< control>-V	2710
< control>-W	2710
< cpio.h>	227
< ctype.h>	229
< dirent.h>	231
< dlfcn.h>	233
< dollar-sign>	54
< errno.h>	234
< fcntl.h>	238
< fenv.h>	243
< oat.h>	247
< fmtmsg.h>	252
< fnmatch.h>	254
< form-feed>	63
< ftw.h>	255
< glob.h>	257
< grp.h>	259
< iconv.h>	261
< inttypes.h>	262
< iso646.h>	265
< langinfo.h>	266
< libgen.h>	269
< limits.h>	270
< locale.h>	285
< math.h>	288
< monetary.h>	296
< mqueue.h>	297
< ndbm.h>	299
< net/if.h>	301

<netdb.h>	302
<netinet/in.h>	306
<netinet/tcp.h>	311
<newline>.....	72
<nl_types.h>	312
<number-sign>	73
<period>	77
<poll.h>	313
<pthread.h>	315, 3644
<pwd.h>	321
<regex.h>.....	323
<sched.h>	325
<search.h>	327
<semaphore.h>.....	329
<setjmp.h>.....	331
<signal.h>.....	332
<slash>.....	91
<space>	92
<spawn.h>.....	341
<stdarg.h>	343
<stdbool.h>	345
<stddef.h>	346
<stdint.h>	348
<stdio.h>.....	355
<stdlib.h>	359
<string.h>	363
<strings.h>	365
<stropts.h>	366
<sys/dir.h>	231
<sys/ipc.h>	371
<sys/mman.h>	373
<sys/msg.h>	376
<sys/resource.h>	378
<sys/select.h>.....	380
<sys/sem.h>	382
<sys/shm.h>	384
<sys/socket.h>	386
<sys/stat.h>	392
<sys/statvfs.h>.....	397
<sys/time.h>	399
<sys/times.h>.....	401
<sys/types.h>.....	402
<sys/uio.h>.....	406
<sys/un.h>.....	407
<sys/utsname.h>	408
<sys/wait.h>.....	409
<syslog.h>	411
<tab>	98
<tar.h>.....	413
<termios.h>	415
<tgmath.h>.....	421
<tilde>.....	99

<time.h>	425
<trace.h>	429
<ulimit.h>	433
<unistd.h>	434
<utime.h>	455
<utmpx.h>	456
<vertical-tab>	104
<wchar.h>	458
<wctype.h>	463
<wordexp.h>	465
±0	106
_asm_builtin_atoi()	3566
_BSD	3689
_CFLAGS	2547
_Complex_I	224
_CS_PATH	441
_CS_POSIX_V6_ILP32_OFF32_CFLAGS	443
_CS_POSIX_V6_ILP32_OFF32_LDFLAGS	443
_CS_POSIX_V6_ILP32_OFF32_LIBS	443
_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS	443
_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS	443
_CS_POSIX_V6_ILP32_OFFBIG_LIBS	443
_CS_POSIX_V6_LP64_OFF64_CFLAGS	443
_CS_POSIX_V6_LP64_OFF64_LDFLAGS	443
_CS_POSIX_V6_LP64_OFF64_LIBS	443
_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS	443
_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS	443
_CS_POSIX_V6_LPBIG_OFFBIG_LIBS	443
_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS	443
_CS_POSIX_V7_ILP32_OFF32_CFLAGS	441
_CS_POSIX_V7_ILP32_OFF32_LDFLAGS	442
_CS_POSIX_V7_ILP32_OFF32_LIBS	442
_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS	442
_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS	442
_CS_POSIX_V7_ILP32_OFFBIG_LIBS	442
_CS_POSIX_V7_LP64_OFF64_CFLAGS	442
_CS_POSIX_V7_LP64_OFF64_LDFLAGS	442
_CS_POSIX_V7_LP64_OFF64_LIBS	442
_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS	442
_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS	442
_CS_POSIX_V7_LPBIG_OFFBIG_LIBS	443
_CS_POSIX_V7_THREADS_CFLAGS	443
_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS	443
_CS_V6_ENV	443
_CS_V7_ENV	443
_Exit()	553
_exit()	553, 2232, 3583, 3601
_Exit()	3768
_exit()	3768
_FILE	608
_Imaginary_I	224
_IOFBF	355, 1888, 1931

_IOLBF	355, 875, 1931
_IONBF	355, 1888, 1931
_LDFLAGS	2547
_LIBS	2547
LINE	608
_longjmp()	559, 3771
_LVL	476
_MAX	475
_MIN	270, 475
_PC constants	
defined in <unistd.h>	444
used in pathconf	902
_PC_2_SYMLINKS	902
_PC_ALLOC_SIZE_MIN	902
_PC_ASYNC_IO	902
_PC_CHOWN_RESTRICTED	902
_PC_FILESIZEBITS	902
_PC_LINK_MAX	902
_PC_MAX_CANON	902
_PC_MAX_INPUT	902
_PC_NAME_MAX	902
_PC_NO_TRUNC	902
_PC_PATH_MAX	902
_PC_PIPE_BUF	902
_PC_PRIO_IO	902
_PC_REC_INCR_XFER_SIZE	902
_PC_REC_MAX_XFER_SIZE	902
_PC_REC_MIN_XFER_SIZE	902
_PC_REC_XFER_ALIGN	902
_PC_SYMLINK_MAX	902
_PC_SYNC_IO	902
_PC_TIMESTAMP_RESOLUTION	902
_PC_VDISABLE	902
_POSIX	270
_POSIX maximum values	
in <limits.h>	275
_POSIX minimum values	
in <limits.h>	276
_POSIX2 constants	
in sysconf	2099
_POSIX2_BC_BASE_MAX	274, 279
_POSIX2_BC_DIM_MAX	275, 279
_POSIX2_BC_SCALE_MAX	275, 279
_POSIX2_BC_STRING_MAX	275, 279
_POSIX2_CHARCLASS_NAME_MAX	275, 279
_POSIX2_CHAR_TERM	439, 2101
_POSIX2_COLL_WEIGHTS_MAX	275, 279
_POSIX2_C_BIND	17, 439, 2101
_POSIX2_C_DEV	439, 2101
_POSIX2_EXPR_NEST_MAX	275, 279
_POSIX2_FORT_DEV	439, 2101
_POSIX2_FORT_RUN	439, 2101

_POSIX2_LINE_MAX.....	275, 279, 282
_POSIX2_LOCALEDEF.....	439, 2101
_POSIX2_PBS.....	439, 2101
_POSIX2_PBS_ACCOUNTING.....	439, 2101
_POSIX2_PBS_CHECKPOINT.....	440, 2101
_POSIX2_PBS_LOCATE.....	440, 2101
_POSIX2_PBS_MESSAGE.....	440, 2101
_POSIX2_PBS_TRACK.....	440, 2101
_POSIX2_RE_DUP_MAX.....	279
_POSIX2_SW_DEV.....	440, 2101
_POSIX2_SYMLINKS.....	441
_POSIX2_UPE.....	440, 2101
_POSIX2_VERSION.....	434, 2101
_POSIX.....	474
_POSIX_ADVISORY_INFO.....	18, 23, 434, 904, 2100, 3777
_POSIX_AIO_LISTIO_MAX.....	270, 276
_POSIX_AIO_MAX.....	271, 276
_POSIX_ARG_MAX.....	271, 276
_POSIX_ASYNCHRONOUS_IO.....	17, 435, 2100, 3485, 3777
_POSIX_ASYNC_IO.....	441, 902
_POSIX_BARRIERS.....	17, 435, 2100, 3485, 3778
_POSIX_CHILD_MAX.....	271, 276
_POSIX_CHOWN_RESTRICTED.....	17, 435, 670, 902, 905, 3479, 3778
_POSIX_CLOCKRES_MIN.....	275
_POSIX_CLOCK_SELECTION.....	17, 435, 2100, 3485, 3778
_POSIX_CPUTIME.....	18, 23, 435, 2100, 3778
_POSIX_C_SOURCE.....	472-473, 3567, 3570
_POSIX_DELAYTIMER_MAX.....	271, 276
_POSIX_FSYNC.....	18, 20, 23, 435, 2100, 3778
_POSIX_HOST_NAME_MAX.....	271, 276
_POSIX_IPV6.....	18, 435, 2100, 3778
_POSIX_JOB_CONTROL.....	17, 435, 2100, 3479, 3778, 3782
_POSIX_LINK_MAX.....	273, 276
_POSIX_LOGIN_NAME_MAX.....	271, 276
_POSIX_MAPPED_FILES.....	17, 435, 2100, 3485, 3778
_POSIX_MAX_CANON.....	273, 276
_POSIX_MAX_INPUT.....	274, 276
_POSIX_MEMLOCK.....	18, 23, 435, 2100, 3779
_POSIX_MEMLOCK_RANGE.....	18, 23, 435, 2100, 3779
_POSIX_MEMORY_PROTECTION.....	17, 435, 2100, 3485, 3779
_POSIX_MESSAGE_PASSING.....	18, 23, 435, 2100, 3779
_POSIX_MONOTONIC_CLOCK.....	18, 23, 436, 2100, 3779
_POSIX_MQ_OPEN_MAX.....	271, 276
_POSIX_MQ_PRIO_MAX.....	271, 276
_POSIX_NAME_MAX.....	274, 277, 1358, 1368, 1546, 1854, 1864, 1938
_POSIX_NGROUPS_MAX.....	275, 277
_POSIX_NO_TRUNC.....	17, 112, 436, 902, 3479
_POSIX_OPEN_MAX.....	271, 277, 1088
_POSIX_PATH_MAX.....	274, 277, 407, 1358, 1368, 1854, 1864, 1938
_POSIX_PIPE_BUF.....	274, 277
_POSIX_PRIORITIZED_IO.....	18, 23, 436, 504, 2100, 3779
_POSIX_PRIORITY_SCHEDULING.....	18, 23-24, 436, 504, 2100, 3779

_POSIX_PRIO_IO.....	441, 902
_POSIX_RAW_SOCKETS	18, 436, 2100
_POSIX_READER_WRITER_LOCKS	17, 436, 2100, 3485
_POSIX_REALTIME_SIGNALS.....	17, 436, 2100, 3485, 3779
_POSIX_REGEX.....	17, 436, 2100, 3779
_POSIX_RE_DUP_MAX.....	275, 277
_POSIX_RTSIG_MAX.....	272, 277, 3577, 3784
_POSIX_SAVED_IDS.....	17, 436, 2100, 3479, 3780
_POSIX_SEMAPHORES	17, 436, 2100, 3485, 3780
_POSIX_SEM_NSEMS_MAX	272, 277
_POSIX_SEM_VALUE_MAX	272, 277
_POSIX_SHARED_MEMORY_OBJECTS	18, 23, 436, 2100, 3780
_POSIX_SHELL	17, 436, 2100, 3780
_POSIX_SIGQUEUE_MAX.....	272, 277
_POSIX_SOURCE.....	473, 3567
_POSIX_SPAWN.....	18, 23, 436, 2100, 3780
_POSIX_SPINLOCKS	3780
_POSIX_SPIN_LOCKS	17, 437, 2100, 3485
_POSIX_SPORADIC_SERVER.....	18, 23-24, 437, 2100, 3780
_POSIX_SSIZE_MAX.....	277, 281
_POSIX_SS_REPL_MAX	272, 277, 2100, 3617
_POSIX_STREAM_MAX.....	272, 277
_POSIX_SYMLINK_MAX.....	274, 278
_POSIX_SYMLOOP_MAX.....	272, 278
_POSIX_SYNCHRONIZED_IO	18, 23, 437, 2100, 3780
_POSIX_SYNC_IO	441, 902, 3778
_POSIX_THREADS.....	17, 438, 2100, 3485, 3780
_POSIX_THREAD_ATTR_STACKADDR	18, 20, 437, 2100, 3780
_POSIX_THREAD_ATTR_STACKSIZE.....	18, 20, 437, 2100, 3781
_POSIX_THREAD_CPUTIME	18, 24-25, 437, 2100
_POSIX_THREAD_DESTRUCTOR_ITERATIONS.....	272, 278
_POSIX_THREAD_KEYS_MAX	272, 278
_POSIX_THREAD_PRIORITY_SCHEDULING.....	18, 24-25, 437, 2100, 3781
_POSIX_THREAD_PRIO_INHERIT	18, 24, 437, 2100, 3781
_POSIX_THREAD_PRIO_PROTECT	18, 24, 437, 2100, 3781
_POSIX_THREAD_PROCESS_SHARED	18, 20, 437, 1681, 2100, 3781
_POSIX_THREAD_ROBUST_PRIO_INHERIT	24, 437, 2100
_POSIX_THREAD_ROBUST_PRIO_PROTECT	24, 437, 2100
_POSIX_THREAD_SAFE_FUNCTIONS.....	17, 438, 2100, 3485, 3781
_POSIX_THREAD_SPORADIC_SERVER.....	18, 24-25, 438, 2100, 3781
_POSIX_THREAD_THREADS_MAX	272, 278
_POSIX_TIMEOUTS.....	17, 438, 2100, 3485, 3781
_POSIX_TIMERS	17, 438, 2101, 3485, 3781
_POSIX_TIMER_MAX.....	272, 278
_POSIX_TIMESTAMP_RESOLUTION	441, 902
_POSIX_TRACE	18, 25, 438, 2101, 3781
_POSIX_TRACE_EVENT_FILTER.....	18, 25, 438, 2101, 3782
_POSIX_TRACE_EVENT_NAME_MAX.....	273, 278, 1517, 1519, 2101
_POSIX_TRACE_INHERIT	18, 25, 438, 2101, 3782
_POSIX_TRACE_LOG.....	18, 25, 438, 2101, 3782
_POSIX_TRACE_NAME_MAX	273, 278, 2101
_POSIX_TRACE_SYS_MAX.....	273, 278, 1514, 2101

_POSIX_TRACE_USER_EVENT_MAX	273, 278, 1519, 2101
_POSIX_TTY_NAME_MAX	273, 278
_POSIX_TYPED_MEMORY_OBJECTS	18, 23, 438, 2101, 3782
_POSIX_TZNAME_MAX	273, 278, 3539
_POSIX_V6_ILP32_OFF32	438, 2101
_POSIX_V6_ILP32_OFFBIG	438, 2101
_POSIX_V6_LP64_OFF64	438, 2101
_POSIX_V6_LPBIG_OFFBIG	439, 2101
_POSIX_V7_ILP32_OFF32	439, 2101
_POSIX_V7_ILP32_OFFBIG	439, 2101
_POSIX_V7_LP64_OFF64	439, 2101
_POSIX_V7_LPBIG_OFFBIG	439, 2101
_POSIX_VDISABLE	18, 447, 902, 3271, 3479
_POSIX_VERSION	17, 434, 2101, 2191
_PROCESS	476
_PTHREAD_THREADS_MAX	1649
_SC constants	
defined in <unistd.h>	444
in sysconf	2099
_SC_2_CHAR_TERM	2101
_SC_2_C_BIND	2101
_SC_2_C_DEV	2101
_SC_2_FORT_DEV	2101
_SC_2_FORT_RUN	2101
_SC_2_LOCALEDEF	2101
_SC_2_PBS_ACCOUNTING	2101
_SC_2_PBS_CHECKPOINT	2101
_SC_2_PBS_LOCATE	2101
_SC_2_PBS_MESSAGE	2101
_SC_2_PBS_TRACK	2101
_SC_2_SW_DEV	2101
_SC_2_UPE	2101
_SC_2_VERSION	1438, 2101
_SC_ADVISORY_INFO	2100
_SC_AIO_LISTIO_MAX	2099
_SC_AIO_MAX	2099
_SC_AIO_PRIO_DELTA_MAX	2099
_SC_ARG_MAX	2099
_SC_ASYNCHRONOUS_IO	2100
_SC_ATEXIT_MAX	2099
_SC_BARRIERS	2100
_SC_BC_BASE_MAX	2099
_SC_BC_DIM_MAX	2099
_SC_BC_SCALE_MAX	2099
_SC_BC_STRING_MAX	2099
_SC_CHILD_MAX	2099
_SC_CLK_TCK	2099, 2159
_SC_CLOCK_SELECTION	2100
_SC_COLL_WEIGHTS_MAX	2099
_SC_CPU_TIME	2100
_SC_DELAYTIMER_MAX	2099
_SC_EXPR_NEST_MAX	2099

_SC_FSYNC	2100
_SC_GETGR_R_SIZE_MAX	1035, 2099
_SC_GETPW_R_SIZE_MAX	2099
_SC_IOV_MAX	2099
_SC_IPV6	2100
_SC_JOB_CONTROL	2100
_SC_LINE_MAX	2099
_SC_LOGIN_NAME_MAX	2099
_SC_MEMLOCK	2100
_SC_MEMLOCK_RANGE	2100
_SC_MEMORY_PROTECTION	2100
_SC_MESSAGE_PASSING	2100
_SC_MONOTONIC_CLOCK	2100
_SC_MQ_OPEN_MAX	2099
_SC_MQ_PRIO_MAX	2099
_SC_NGROUPS_MAX	2099
_SC_OPEN_MAX	2099
_SC_PAGESIZE	1444, 2099, 3601-3602
_SC_PAGE_SIZE	2099
_SC_PRIORITIZED_IO	2100
_SC_PRIORITY_SCHEDULING	2100
_SC_RAW_SOCKETS	2100
_SC_READER_WRITER_LOCKS	2100
_SC_REALTIME_SIGNALS	2100
_SC_REGEX	2100
_SC_RE_DUP_MAX	2099
_SC_RTSIG_MAX	2099
_SC_SAVED_IDS	2100
_SC_SEMAPHORES	2100
_SC_SEM_NSEMS_MAX	2100
_SC_SEM_VALUE_MAX	2100
_SC_SHARED_MEMORY_OBJECTS	2100
_SC_SHELL	2100
_SC_SIGQUEUE_MAX	2100
_SC_SPAWN	2100
_SC_SPIN_LOCKS	2100
_SC_SPORADIC_SERVER	2100
_SC_SS_REPL_MAX	2100
_SC_STREAM_MAX	2100
_SC_SYMLOOP_MAX	2100
_SC_SYNCHRONIZED_IO	2100
_SC_THREADS	2100
_SC_THREAD_ATTR_STACKADDR	2100
_SC_THREAD_ATTR_STACKSIZE	2100
_SC_THREAD_CPUTIME	2100
_SC_THREAD_DESTRUCTOR_ITERATIONS	2099
_SC_THREAD_KEYS_MAX	2099
_SC_THREAD_PRIORITY_SCHEDULING	2100
_SC_THREAD_PRIO_INHERIT	2100
_SC_THREAD_PRIO_PROTECT	2100
_SC_THREAD_PROCESS_SHARED	2100
_SC_THREAD_ROBUST_PRIO_INHERIT	2100

_SC_THREAD_ROBUST_PRIO_PROTECT.....	2100
_SC_THREAD_SAFE_FUNCTIONS.....	2100
_SC_THREAD_SPORADIC_SERVER.....	2100
_SC_THREAD_STACK_MIN.....	2099
_SC_THREAD_THREADS_MAX.....	2099
_SC_TIMEOUTS.....	2100
_SC_TIMERS.....	2101
_SC_TIMER_MAX.....	2100
_SC_TRACE.....	2101
_SC_TRACE_EVENT_FILTER.....	2101
_SC_TRACE_EVENT_NAME_MAX.....	2101
_SC_TRACE_INHERIT.....	2101
_SC_TRACE_LOG.....	2101
_SC_TRACE_NAME_MAX.....	2101
_SC_TRACE_SYS_MAX.....	2101
_SC_TRACE_USER_EVENT_MAX.....	2101
_SC_TTY_NAME_MAX.....	2100
_SC_TYPED_MEMORY_OBJECTS.....	2101
_SC_TZNAME_MAX.....	2100
_SC_V6_ILP32_OFF32.....	2101
_SC_V6_ILP32_OFFBIG.....	2101
_SC_V6_LP64_OFF64.....	2101
_SC_V6_LPBIG_OFFBIG.....	2101
_SC_V7_ILP32_OFF32.....	2101
_SC_V7_ILP32_OFFBIG.....	2101
_SC_V7_LP64_OFF64.....	2101
_SC_V7_LPBIG_OFFBIG.....	2101
_SC_VERSION.....	2101
_SC_XOPEN_CRYPT.....	2101
_SC_XOPEN_ENH_I18N.....	2101
_SC_XOPEN_REALTIME.....	2101
_SC_XOPEN_REALTIME_THREADS.....	2101
_SC_XOPEN_SHM.....	2101
_SC_XOPEN_STREAMS.....	2101
_SC_XOPEN_UNIX.....	2101
_SC_XOPEN_UUCP.....	2101
_SC_XOPEN_VERSION.....	2101
_setjmp().....	559, 3771
_t.....	476
_TIME.....	476
_tolower().....	561
_toupper().....	562
_XOPEN_CRYPT.....	18, 22, 440, 2101
_XOPEN_ENH_I18N.....	440, 2101
_XOPEN_IOV_MAX.....	271, 279
_XOPEN_NAME_MAX.....	274, 279, 1358, 1368, 1547, 1854, 1864, 1938
_XOPEN_PATH_MAX.....	274, 279, 1358, 1368, 1546, 1854, 1864, 1938
_XOPEN_REALTIME.....	18, 22-23, 440, 829, 2101
_XOPEN_REALTIME_THREADS.....	18, 24, 440, 2101
_XOPEN_SHM.....	440, 2101
_XOPEN_SOURCE.....	473, 3567
_XOPEN_STREAMS.....	18, 25-26, 440, 2101

_XOPEN_UNIX.....	18-19, 440, 2101
_XOPEN_UUCP.....	440, 2101
_XOPEN_VERSION.....	19, 434, 2101
__errno().....	3574
a64l().....	563
ABDAY_.....	267
ABDAY_1.....	1403
ABMON_.....	267
abort().....	565 , 3768
abortive release.....	33
abs().....	567
absolute pathname.....	33, 111
accept().....	568
access mode.....	33
access().....	570 , 3511, 3768
Account_Name.....	2435
acos().....	574
acosf().....	574
acosh().....	576
acoshf().....	576
acoshl().....	576
acosl().....	574, 578
ACTION.....	1117
actions equivalent to functions.....	2331
active trace stream.....	3688
adb	
rationale for omission.....	3756
additional file access control mechanism.....	33
address families.....	3661
address information.....	934
address space.....	33
address string.....	934
addressing.....	3662
addrinfo.....	303 , 934
admin.....	2454
ADV.....	7
advanced realtime.....	23
ADVANCED REALTIME.....	341, 678, 1440, 1442, 1444, 1446, 1448, 1452, 1460, 1463, 1466, 1468
.....	1470, 1472, 1474, 1476, 1478, 1480, 1543, 1545
advanced realtime threads.....	24
ADVANCED REALTIME THREADS.....	1643
advisory information.....	34, 3587
affirmative response.....	34
AF.....	476
AF_INET.....	388
AF_INET6.....	388
AF_UNIX.....	388
AF_UNSPEC.....	389, 702
AIO.....	475
aio.....	475
AIO_ALLDONE.....	220, 579
aio_cancel().....	579 , 3596

AIO_CANCELED	220, 579
aio_error()	581
aio_fsync()	583 , 3580, 3595
AIO_LISTIO_MAX	270, 1249, 2099, 3782
AIO_MAX	271, 1249, 2099, 3782
AIO_NOTCANCELED	220, 579
AIO_PRIO_DELTA_MAX	271, 504, 2099, 3782
aio_read()	585 , 3596
aio_return()	588
aio_suspend()	590 , 3595, 3621
aio_write()	592 , 3596
ai_	475
AI_ADDRCONFIG	303, 935
AI_ALL	303, 935
AI_CANONNAME	303, 935
AI_INET6	935
AI_NUMERICHOST	303, 935
AI_NUMERICSERV	303, 935
AI_PASSIVE	303, 935
AI_V4MAPPED	303, 935
alarm()	595 , 3585, 3620, 3768
alert	34
alert character	34
alias	2344, 2367, 2459, 3717, 3772
alias name	34
alias substitution	2348, 3720
alignment	34
alphasort()	597
alternate file access control mechanism	35
alternate signal stack	35
ALT_DIGITS	267
AM_STR	267
anchoring	188
ancillary data	35
AND list	2370, 3742
AND-OR list	2369
angle brackets	35
anycast	533
ANYMARK	369, 1153
API	36
apostrophe character	35
appending redirected output	2361
application	36
application address	36
application conformance	29
application instrumentation	3676
application program interface	36
application-managed thread stack	522, 3661
appropriate privileges	36, 572, 903, 3488
ar.	2462 , 3774-3775
arbitrary file size	3716

archives	
ar command	2462
AREGTYPE	413
argc	791
argument	36
ARG_MAX	271, 481, 784, 788, 793, 2099, 3452, 3498, 3710, 3782
arithmetic expansion	2358, 3732
arithmetic language	
bc.....	2524
arithmetic precision and operations.....	2331
arm (a timer)	36
array identifiers	2529
as	
rationale for omission.....	3756
asa	2470 , 3772, 3774, 3776
ASCII.....	3500
asctime().....	600
asctime_r()	600
asin().....	603
asinf().....	603
asinh().....	605
asinhf()	605
asinhf_l().....	605
asinl()	603, 607
assert()	608
asterisk.....	37
async-cancel safety.....	3659
async-cancel-safe function	37
async-signal-safe	1560, 3583
async-signal-safe function.....	37
asynchronous error	3663
asynchronous events	37
asynchronous I/O.....	3595, 3769
completion.....	37
operation.....	38
asynchronous input and output.....	37
asynchronous lists.....	2370, 3742
asynchronously-generated signal	37
at.....	2473 , 3772
at-job.....	2473
atan().....	609
atan2().....	611
atan2f().....	611
atan2l().....	611
atanf()	609, 614
atanh()	615
atanhf()	615
atanhl()	615
atanl()	609, 617
atexit()	618 , 3659
ATEXIT_MAX.....	271, 618, 2099
atof()	620

atoi()	621, 3565-3566
atol()	623
atoll()	623
attributes	
clock-resolution	542, 1492
creation-time	542, 1492
generation-version	542, 1492
inheritance	542, 1494
log-full-policy	540, 542, 1494, 1497
log-max-size	542, 1495, 1497
max-data-size	542, 1497
stream-full-policy	539-540, 542, 1495
stream-min-size	542, 1498
trace-name	542, 1492
truncation-status	1517
AT_EACCESS	239
AT_FDCWD	239, 570, 665, 671, 966, 987, 1243, 1316, 1322, 1327, 1411, 1783, 1817, 2095 3301
AT_REMOVEDIR	240, 2197
AT_SYMLINK_FOLLOW	240, 1243
AT_SYMLINK_NOFOLLOW	240, 665, 671, 966, 988
authentication	38
authorization	38
automatic storage class	2533
awk	2482, 3772, 3774
actions	2494
arithmetic functions	2496
escape sequences	2492
expression patterns	2494
expressions	2485
functions	2496
grammar	2500
input/output and general functions	2498
lexical conventions	2506
output statements	2495
overall program structure	2485
pattern ranges	2494
patterns	2493
regular expressions	2491
special patterns	2493
string functions	2497
user-defined functions	2499
variables and special variables	2489
background	1909, 3494-3497, 3551
background job	38
background process	38, 2132
background process group	38
background work	
at	2473
batch	2521
bg	2539
crontab	2613

fg	2786
jobs.....	2870
nice	3027
nohup.....	3040
renice.....	3194
backquote	38
BACKREF.....	192
backslash	38, 3718
backspace character	39
bandinfo.....	366
banner	
rationale for omission.....	3757
barrier.....	39, 3637
basename	39, 2518, 3771, 3773
basename().....	624
basic regular expression	39, 183, 3542
batch	2521 , 3772
general concepts	3753
batch access list.....	39
batch administration.....	2430
batch administrator.....	39
batch authorization	2430
batch client	40
batch client-server interaction	2427
batch destination	40
batch destination identifier	40
batch directive.....	40
batch environment	3749
option definitions	3751
services.....	2427
utilities	2427
utilities, common behavior	3756
batch job.....	40
batch job abort	2430, 2441
batch job attribute	41
batch job creation	2428
batch job execution.....	2429, 2433
batch job exit	2430, 2440
batch job identifier	41, 2449
batch job message request.....	2443
batch job name.....	41
batch job owner	41
batch job priority	41
batch job routing.....	2429, 2440
batch job state	41
batch job states.....	2432
batch job status request	2444
batch job tracking	2428
batch name service.....	41
batch name space	42
batch node	42
batch notification.....	2431

batch operator.....	42
batch queue.....	42, 2428
batch queue attribute.....	42
batch queue position.....	42
batch queue priority.....	43
batch queue status request.....	2446
batch rerunability.....	43
batch restart.....	43
batch server.....	43
batch server name.....	43
batch server restart.....	2441
batch service.....	43
batch service request.....	44
batch services.....	2431, 3755
batch submission.....	44
batch system.....	44
historical implementations.....	3750
history.....	3750
batch target user.....	44
batch user.....	44
baud rate functions.....	658
bc.....	2524, 3771-3772, 3777
grammar.....	2525
lexical conventions.....	2527
operations.....	2529
operators.....	2529
bcc (mailer blind carbon copy).....	2966
bcmp().....	3691
bcopy().....	3691
BC_ constants	
in sysconf.....	2099
BC_BASE_MAX.....	274, 2099, 2334, 3710
BC_DIM_MAX.....	275, 2099, 2334, 3710
BC_SCALE_MAX.....	275, 2099, 2334, 3710
BC_STRING_MAX.....	275, 2099, 2334, 2527
BE.....	7
bg.....	2344, 2367, 2539, 3717, 3772
binary primaries.....	3290
bind.....	44
bind().....	626
bi.....	475
blank character.....	44
blank line.....	45
blkcnt_t.....	402
blksize_t.....	402
BLKTYPE.....	413
block special file.....	45
block-mode terminal.....	45
blocked process (or thread).....	45
blocking.....	45
BOOT_TIME.....	456, 771-772
bounded response.....	3770

braces	45
bracket expression	
grammar	3547
brackets.....	45
BRE	
expression anchoring.....	3544
grammar lexical conventions	3547
matching a collating element.....	3542
matching a single character	3542
matching multiple characters.....	3544
ordinary character.....	3542
periods	3542
precedence.....	3544
special character	3542
BRE (ERE) matching a single character	182
BRE (ERE) matching multiple characters	182
break.....	2386
BRKINT	416
broadcast	46
BSD.....	3491, 3550, 3575
BSDLY	417
bsd_signal()	3692
bsearch().....	629
BSn.....	417
btowc()	632
buffer cache.....	974
BUFSIZ.....	355, 1888
built-in	46
built-in utilities	46, 2344, 3716
builtin.....	2600
BUS_.....	475
BUS_ADRALN	337
BUS_ADRERR	337
BUS_OBJERR	337
byte.....	46
byte input/output functions	46
byte-oriented stream.....	498
byte-stream mode.....	1772
bzero()	3692
C Shell.....	3494-3496
C-language extensions.....	3765, 3771
c99.....	2542 , 3774
external symbols.....	2546
standard libraries	2545
cabs().....	633
cabsf()	633
cabsl()	633
cacos().....	634
cacosf()	634
cacosh()	635
cacoshf()	635
cacoshl()	635

cacosl()	634, 636
cal	2554
calendar	
rationale for omission	3757
calloc()	637
can	5
cancel	
rationale for omission	3757
cancel-safe	1728
cancelability state	517, 1650, 1728
cancelability type	1650, 1728
canceling execution of a thread	1603
cancellation cleanup handler	1608, 1620, 1639, 1653, 3657-3658
cancellation cleanup stack	3657
cancellation points	518
canonical mode input processing	202, 3551
canonical name	935
carg()	639
cargf()	639
cargl()	639
carriage-control characters	2470
carriage-return character	46
case	3743
case conditional construct	2372
case folding	3512-3513
casin()	640
casinf()	640
casinh()	641
casinhf()	641
casinhl()	641
casinl()	640, 642
cat	2557, 3716
catan()	643
catanf()	643
catanh()	644
catanhf()	644
catanhl()	644
catanl()	643, 645
catclose()	646, 3773
catgets()	647, 3773
catopen()	649, 3773
CBAUD	477
cbrt()	651
cbrtf()	651
cbrtl()	651
cc (mailer carbon copy)	2966
ccos()	652
ccosf()	652
ccosh()	653
ccoshf()	653
ccoshl()	653
ccosl()	652, 654

CD.....	7
cd	2344, 2367, 2561, 3717, 3773
ceil()	655
ceilf().....	655
ceill().....	655
CEO	3543
cexp()	657
cexpf().....	657
cexpl().....	657
cfgetispeed()	658
cfgetospeed()	660
c ow	2566
cfsetispeed().....	661
cfsetospeed().....	662
change current working directory	664, 2331
change file modes.....	668
change history.....	3480, 3561, 3707
change owner and group of file	672
char	548, 3691
character	47, 3489
rationale.....	3489
character array	47
character class	47
character counting.....	3434
character encoding.....	128, 3522
state-dependent.....	133
character set	47, 3521
description file	3522
portable filename.....	3500
character special file.....	47
character string.....	47
CHARCLASS_NAME_MAX.....	275, 3528
charmap	
description.....	129
with localedef.....	2909
writing names with locale.....	2903
charmap file	2907, 3271
CHAR_BIT	280
CHAR_MAX	280, 1260, 1262, 3530
CHAR_MIN	280
chdir().....	663
Checkpoint	2435
chgrp	2569, 3716, 3773-3774
child process.....	48, 3489
CHILD_MAX	271, 899, 2099, 3479, 3690, 3710, 3742, 3782
chmod	2572, 3716, 3773-3774
grammar	2575
chmod()	665, 3768
chown	2579, 3716, 3773-3774
chown()	670, 3768
chroot	
rationale for omission.....	3757

chroot()	3501
CHRTYPE	413
cimag()	674
cimagf()	674
cimagl()	674
circum ex	48
cksum	2583, 3716, 3773
CLD_	475
CLD_CONTINUED	337
CLD_DUMPED	337
CLD_EXITED	337
CLD_KILLED	337
CLD_STOPPED	337
CLD_TRAPPED	337
clearerr()	675
CLOCAL	418
clock	48, 3617
clock jump	48
clock tick	48, 595, 2102, 2159, 3489
per second	2099
rationale	3489
clock()	676
clock-resolution attribute	542, 1492
clockid_t	402
CLOCKRES_MIN	3782
clocks	3617
CLOCKS_PER_SEC	402, 425, 676
CLOCK_	476
clock_	476
clock_getcpuclockid()	678 , 3623, 3625
clock_getres()	679
clock_gettime()	679
CLOCK_MONOTONIC	426, 511, 684
clock_nanosleep()	683 , 3621-3622
CLOCK_PROCESS_CPUTIME_ID	426, 512, 3623, 3625
CLOCK_REALTIME	275, 426, 511, 679, 684, 1389, 1676, 2151, 3617-3621
clock_settime()	679, 686
clock_t	402
CLOCK_THREAD_CPUTIME_ID	426, 512, 3623, 3625
clog()	687
clogf()	687
clogl()	687
close a file	690
close()	688 , 3601, 3768
closedir()	692 , 3769
closelog()	694 , 3774
cmp	2588 , 3716, 3772
msg_	476
MSG_	477
MSG_DATA	387
MSG_FIRSTHDR	387
MSG_NXTHDR	387

coded character set.....	48
codes.....	3482
codeset	49
CODESET	267
codeset conversion	2853
tr.....	3310
col	
rationale for omission.....	3757
collating element	49
collating element order.....	3543
collation	49
collation sequence	49
COLL_ELEM_MULTI.....	192
COLL_ELEM_SINGLE	192
COLL_WEIGHTS_MAX	275, 2099, 2334, 3710
colon	2389
column position.....	50, 3490
COLUMNS.....	177 , 3538
comm.....	2592 , 3772
command.....	50, 2344, 2367, 2596, 3490, 3717, 3771
command execution.....	3739
command interpreter	
portable.....	2232
command language.....	3765, 3771
command language interpreter.....	50
command mode.....	2677
command search.....	2367, 3739
command substitution.....	2357, 3730
communications commands	
mailx.....	2943
talk.....	3281
uucp.....	3353
uudecode	3357
uuencode	3360
uustat	3365
uux.....	3368
write	3444
compare thread IDs.....	1638
compilation environment.....	472 , 3566
compilers	
c99.....	2542
fort77	2810
yacc.....	3454
complex.....	224
complex data manipulation.....	3766, 3772
composite graphic symbol	50
compound commands.....	2371, 3743
compound-list.....	2369
compress.....	2602
compression	
compress.....	2602
uncompress	3337

zcat	3471
concepts	3482
concurrent execution.....	107, 3511
of processes	2327
condition variable	50
conditional construct	
case	3743
if	3744
configurable limits	3777, 3782
configuration interrogation	3764, 3767
configuration options	3775
shell and utilities	3775
system interfaces	3777
configuration values	2831
conformance	15, 29, 3480, 3483-3484, 3487-3488, 3514, 3689
POSIX.....	15
POSIX system interfaces.....	17
XSI	15
XSI system interfaces	19
conformance document.....	16, 3480
rationale.....	3480
conforming application	16, 1999, 2467, 3487, 3576, 3713, 3715
conforming application, strictly.....	595, 791, 3484, 3487, 3582
conforming implementation options	20
confstr()	698 , 3769
conj()	701
conjf().....	701
conjl().....	701
connect().....	702
connected socket.....	51
connection	51
connection indication queue.....	3663
connection mode	51
connectionless mode.....	51
consequences of shell errors	2363
continue	2391
control character.....	51, 3268
control data	500
control mode	3554
control operator	51
control-normal.....	1772
controlling process	51
controlling terminal	52, 200, 2327, 3491, 3550, 3768
CONTTYPE.....	413
conversion descriptor	52, 785, 790, 1123-1124, 1126-1127
conversion specification	909, 949, 993, 1003, 2039
modi ed	2047
conversion speci er	
modi ed	2065
Coordinated Universal Time (UTC)	2639
copy	155

copy	les commands	
cp	2605
dd	2641
ln	2898
mv	3017
pax	3068
copysign()	705
copysignf()	705
copysignl()	705
core	2233, 3512
core le	52, 555
cos()	706
cosf()	706
cosh()	708
coshf()	708
coshl()	708
cosl()	706, 710
covert channel	1227, 3514
cp	2605 , 3716, 3773
cpio		
rationale for omission	3757
cpio format	3089
cpow()	711
cpowf()	711
cpowl()	711
cpp		
rationale for omission	3757
cproj()	712
cprojf()	712
cprojl()	712
CPT	7
CPU	401
CPU time	52, 3299
clock	52
timer	52
CRDLY	416
CREAD	418
creal()	713
crealf()	713
creall()	713
creat()	714 , 3601, 3716
create a per-process timer	2152
create an interprocess channel	1431
create session and set process group ID	1922
creation-time attribute	542, 1492
CRn	416
CRNCYSTR	267
cron daemon	2616
crontab	2613 , 3772
CRYPT	716, 756, 1901
crypt()	716
csin()	718

csinf()	718
csinh()	719
csinhf()	719
csinhl()	719
csinl()	718, 720
CSIZE	418, 3554
CSn	418
csplit	2617 , 3772
csqrt()	721
csqrtf()	721
csqrtl()	721
CSTOPB	418
CS_POSIX_V7_THREADS_LDFLAGS	443
ctags	2621 , 3774
ctan()	722
ctanf()	722
ctanh()	723
ctanhf()	723
ctanhl()	723
ctanl()	722, 724
ctermid()	725
ctime()	727 , 3771
ctime_r()	727
cu	
rationale for omission	3757
currency_symbol	155
current job	52
current working directory	53, 105, 2327
cursor position	53
cut	2626 , 3772
CX	7
cxref	2631
c_	476
C_ constants in <cpio.h>	227
C_IRGRP	227
C_IROTH	227
C_IRUSR	227
C_ISBLK	227
C_ISCHR	227
C_ISCTG	227
C_ISDIR	227
C_ISFIFO	227
C_ISGID	227
C_ISLNK	227
C_ISREG	227
C_ISSOCK	227
C_ISUID	227
C_ISVTX	227
C_IWGRP	227
C_IWOTH	227
C_IWUSR	227
C_IXGRP	227

C_IXOTH.....	227
C_IXUSR.....	227
data access.....	3764, 3768
data key creation.....	1654
data keywords.....	3119
data messages.....	500
data segment.....	53
data structure	
dirent.....	231
entry.....	327
group.....	259
lconv.....	285
msqid_ds.....	376
stat.....	392
data type.....	547, 3689
ACTION.....	327
cc_t.....	415
DIR.....	231
div_t.....	359
ENTRY.....	327
FILE.....	355
fpos_t.....	355
glob_t.....	257
ldiv_t.....	359
lldiv_t.....	359
mbstate_t.....	458
msglen_t.....	376
msgqnum_t.....	376
nl_catd.....	312
nl_item.....	312
pid_t.....	332
ptrdiff_t.....	346
regex_t.....	323
regmatch_t.....	323
regoff_t.....	323
shmatt_t.....	384
sigset_t.....	332
sig_atomic_t.....	332
size_t.....	346
speed_t.....	415
tflag_t.....	415
VISIT.....	327
wchar_t.....	346
wctrans_t.....	463
wctype_t.....	458
wint_t.....	458
data types	
defined in <fcntl.h>.....	243
defined in <sys/types.h>.....	402
date.....	2634, 3773
conversion specifications.....	2634
modified conversion specifications.....	2635

DATEMSK	177, 1020
datum	299
daylight	729, 2185
DAY_	267
DBL_ constants	
defined in <float.h>	248
DBL_DIG	249
DBL_EPSILON	250
DBL_MANT_DIG	248
DBL_MAX	250
DBL_MAX_10_EXP	250
DBL_MAX_EXP	249
DBL_MIN	250, 603, 605, 609, 611, 615, 776, 798, 800, 802, 833, 885
DBL_MIN_10_EXP	249
DBL_MIN_EXP	249
DBM	299, 730-731
DBM_	475
dbm_	475
dbm_clearerr()	730
dbm_close()	730
dbm_delete()	730
dbm_error()	730
dbm_fetch()	730
dbm_rstkey()	730
DBM_INSERT	299, 731
dbm_nextkey()	730
dbm_open()	730
DBM_REPLACE	299, 731
dbm_store()	730
dc	
rationale for omission	3757
dd	2641, 3716, 3772-3773
DEAD_PROCESS	456, 771-772
DECIMAL_DIG	248
default initialization	107
default queue	2445
DEFECHO	477
deferred batch service	53, 2433
deferred cancelability	1650
defined types	547, 3689
definitions	3482
delay process execution	1998
DELAYTIMER_MAX	271, 2099, 2156, 3782
delete batch job request	2442
delta	2650
dependency order	744
descriptive name	934
destination	2450
destroying a mutex	1662
destructor functions	1653
detaching a thread	1636
determinism	3764

device	53
output.....	198
device ID.....	53
device number	3491
device, logical	3497
DEV_BSIZE	395
dev_t.....	402
df.....	2654, 3716, 3773-3774
diff.....	2658, 3772-3773
binary output format	2660
default output format	2660
directory comparison format.....	2659
-c or -C output format.....	2661
-e output format.....	2661
-f output format	2661
-u or -U output format.....	2662
difftime()	734
DIR	231, 547, 692, 1778, 1780, 1822, 1843, 2141
dircmp	
rationale for omission.....	3757
direct I/O.....	3491
directive	909, 949, 993, 1003
directory	53, 3491
device.....	3547
entry	53, 3491
les	3547
link.....	53
list	2923
operations.....	840
protection.....	108, 3511
root	3501
stream.....	54
structure.....	3547
directory commands	
cd	2561
pwd	3130
dirent.....	231, 840
dirfd()	735
dirname.....	2667, 3771, 3773
dirname()	736
DIRTYPE.....	413
dis	
rationale for omission.....	3757
disarm (a timer)	54
disk space commands	
df.....	2654
du.....	2670
ulimit.....	3326
display	54, 3491
display line.....	54
div()	738
dlclose().....	739

derror()	741
dlopen()	743
dlsym()	746
documentation	16, 2992
dollar-sign	54
domain error	117
dot	54, 840, 1819, 2393, 3491
dot-dot	55, 840, 1819, 3492, 3500, 3517
double-quote	55, 2346, 3718
downshifting	55
dprintf()	748, 909
drand48()	749
driver	55
du	2670, 3716, 3773-3774
dup()	752, 3601, 3768
dup2()	752, 3601, 3768
duplicating an input file descriptor	2363
duplicating an output file descriptor	2363
duplocale()	754
DUP_COUNT	192
dynamic package initialization	1702
d_	475
D_FMT	267
D_T_FMT	267
E2BIG	234, 481
EACCES	234, 482
EADDRINUSE	234, 482
EADDRNOTAVAIL	234, 482
EAFNOSUPPORT	234, 482
EAGAIN	234, 482, 487
EAI_AGAIN	304, 1010
EAI_BADFLAGS	304, 1010
EAI_FAIL	304, 1010
EAI_FAMILY	304, 1010
EAI_MEMORY	304, 1010
EAI_NONAME	304, 1010
EAI_OVERFLOW	304, 1010
EAI_SERVICE	304, 1010
EAI_SOCKTYPE	304, 1010
EAI_SYSTEM	304, 1010
EALREADY	234, 482
EBADF	234, 482
EBADMSG	234, 482
EBUSY	234, 482, 3574, 3645
ECANCELED	234, 482, 3572
ECHILD	234, 482
ECHO	418
echo	2674, 3771
ECHOCTL	477
ECHOE	418, 3554
ECHOK	418, 3554
ECHOKE	477

ECHONL	418, 3554
ECHOPRT	477
ECONNABORTED	234, 483
ECONNREFUSED	234, 483
ECONNRESET	234, 483
ecvt()	3692
ed	2677, 3772-3773
addresses	2679
append command	2681
change command	2682
commands	2680
copy command	2687
delete command	2682
edit command	2682
edit without checking command	2682
filename command	2683
global command	2683
global non-matched command	2687
help command	2684
help-mode command	2684
insert command	2684
interactive global command	2683
interactive global not-matched command	2687
join command	2684
line number command	2688
list command	2684
mark command	2684
move command	2685
null command	2688
number command	2685
print command	2685
prompt command	2685
quit command	2685
quit without checking command	2685
read command	2686
regular expressions	2679
shell escape command	2688
substitute command	2686
undo command	2687
write command	2687
EDEADLK	234, 483
EDESTADDRREQ	234, 483
edit buffer	2697, 3375
edit line	3231
editors	
ed	2677
ex	2697
sed	3216
vi	3375
EDOM	234, 483, 3574
EDQUOT	234, 483
ED_FILE_MAX	2689

ED_LINE_MAX	2690
EEXIST	234, 483
EFAULT	235, 483, 3572
EFBIG	235, 483
effective group ID.....	55, 672, 792, 1042, 2327
effective user ID	55, 572, 792, 1227, 2327, 3511
EFTYPE	3572
EHOSTUNREACH	235, 483
EIDRM	235, 483
eight-bit transparency.....	55
Eighth Edition UNIX	2316, 2600
EILSEQ.....	235, 483, 499, 3574
EINPROGRESS.....	235, 483, 504, 3595
EINTR	235, 484, 520, 3572, 3575, 3585
EINVAL	235, 484, 3572
EIO.....	235, 484
EISCONN	235, 484
EISDIR.....	235, 484
ELOOP	235, 484, 3572
ELSIZE	1290
emacs	
rationale for omission.....	3757
EMFILE	235, 484
EMLINK	235, 484
EMPTY	456, 772
empty directory	56
empty line.....	56
empty string (or null string).....	56
empty wide-character string	56
EMSGSIZE.....	235, 484
EMULTIHOP	235, 484
ENAMETOOLONG.....	235, 484, 3573
encoding	
character	128
encoding rule	56
encrypt().....	756
encryption	22
endgrent()	758, 3510
endhostent().....	760
endnetent()	762
endprotoent()	764
endpwent()	766, 3510
endservent().....	769
endutxent()	771
ENETDOWN	235, 484
ENETRESET	235, 485
ENETUNREACH	235, 485
ENFILE	235, 485
ENOBUFS.....	235, 485
ENODATA	235, 485
ENODEV	235, 485
ENOENT	235, 485

ENOEXEC	235, 485
ENOLCK	235, 485
ENOLINK.....	235, 485
ENOMEM.....	235, 485, 3573
ENOMSG.....	235, 485
ENOPROTOPT.....	235, 485
ENOSPC	235, 485
ENOSR.....	235, 485
ENOSTR	236, 486
ENOSYS.....	236, 486, 3573, 3598
ENOTCONN.....	236, 486
ENOTDIR.....	236, 486
ENOTEMPTY	236, 486
ENOTRECOVERABLE.....	236, 486
ENOTSOCK	236, 486
ENOTSUP.....	236, 486, 3573
ENOTTY.....	236, 486, 3548, 3572-3573
entire regular expression.....	56, 181
ENTRY.....	1117
env	2693 , 3771, 3774
environ.....	774 , 792
environment access.....	3764, 3769
environment variable.....	3537
de nition.....	3537
internationalization.....	174
envp.....	792
ENXIO.....	236, 486
EOF.....	356
EOPNOTSUPP.....	236, 486
E_OVERFLOW	236, 486, 3573
EOWNERDEAD.....	236, 486
EPERM.....	236, 486, 2610, 3654
EPIPE.....	236, 486, 3574
Epoch	57, 3492, 3518, 3617
EPROTO	236, 487
EPROTONOSUPPORT.....	236, 487
EPROTOTYPE	236, 487
equivalence class	57
era	57
ERA	267
erand48().....	749, 775
ERANGE.....	236, 487, 3574
ERASE.....	3551
ERA_D_FMT	267
ERA_D_T_FMT	267
ERA_T_FMT	267
ERE.....	3545
alternation	3546
bracket expression.....	3546
expression anchoring.....	3546
grammar	3547
grammar lexical conventions	3547

matching a collating element.....	3545
matching a single character	3545
matching multiple characters.....	3546
ordinary character	3545
periods	3546
precedence.....	3546
special character	3546
erf()	776
erfc().....	778
erfcf()	778
erfcl()	778
erff()	776, 780
er ().....	776, 780
EROFS.....	236, 487, 3574
errno	781 , 3571
per-thread.....	3574
error conditions	3520, 3737
mathematical functions	117
error descriptions.....	1010
error handling.....	3747
error numbers	481, 3571, 3575
additional	488
Error_Path	2436
escape character.....	3718
escape character (backslash).....	2346
escape sequences	
awk.....	2492
gencat.....	2820
lex	2889
ESPIPE	236, 487
ESRCH	236, 487
EST5EDT.....	2185
establish cancellation handlers.....	1608
establish the locale	2331
ESTALE.....	236, 487
ETIME	236, 487
ETIMEDOUT	236, 487
ETXTBSY	236, 487
eval	2395
event management.....	57
EWOULDBLOCK.....	236, 487
ex.....	2697 , 3772-3773
<backslash>.....	2710
<control>-D command	2733
<newline>.....	2710
abbreviate command	2713
addressing	2703
adjust window command	2731
append command	2713
args command	2714
autoindent option.....	2735
autoprint option	2736

autowrite option.....	2736
beautify option	2736
change command	2714
chdir command	2714
command descriptions	2710
copy command	2714
delete command	2715
directory option.....	2737
edcompatible option	2737
edit command	2715
edit options	2735
errorbells option	2737
escape command	2732
execute command	2734
exrc command.....	2737
file command	2716
global command	2716
ignorecase option	2737
initialization	2701
input editing	2708
insert command.....	2717
join command	2717
list command	2718, 2737
magic command	2738
map command.....	2718
mark command	2720
mesg command	2738
move command.....	2721
next command	2721
number command.....	2722
number option	2738
open command	2722
paragraphs option.....	2738
preserve command.....	2697, 2722
print command	2722
prompt command	2738
put command.....	2723
quit command.....	2723
read command.....	2723
readonly command	2739
recover command.....	2724
redraw command	2739
regular expressions	2734
remap command	2739
replacement strings.....	2735
report command	2739
rewind command	2724
scroll command	2709, 2740
sections command.....	2740
set command.....	2725
shell command.....	2725
shell option.....	2740

shift left command	2733
shift right command	2733
shiftwidth option.....	2740
showmatch option.....	2740
showmode command	2740
slowopen command.....	2741
source command	2725
substitute command	2725
suspend command.....	2727
tabstop option.....	2741
tag command	2727
taglength option	2741
tags command.....	2741
term command	2741
terse command	2741
unabbrev command.....	2728
undo command	2728
unmap command	2728
version command.....	2729
visual command	2729
warn command.....	2742
window command	2742
wrapmargin option.....	2742
wrapscan option.....	2743
write command	2729
write line number command.....	2734
writeany option	2743
xit command	2730
yank command.....	2731
examine and change blocked signals	1736
examine and change signal action	1954
EXDEV	236, 488
exec	783, 2397, 3041
of shell scripts	791
exec family	572, 690, 826, 873, 900, 1560, 1910, 2232, 2344, 2600, 3025, 3452, 3494
.....	3599, 3711, 3745, 3768
exec().....	3682
execl()	783
execle()	783
execlp().....	783
executable file	57
execute	58
execute a file.....	791
execution time.....	52, 58
measurement.....	110, 3514
monitoring.....	58, 512, 3622
Execution_Time	2437
execv()	783
execve()	783
execvp().....	783
EXINIT	2697
exit	2399

exit status.....	3737
and errors	2363
for commands.....	2364
exit().....	796 , 3583, 3768
EXIT_FAILURE.....	359, 553, 796, 3770
EXIT_SUCCESS	359, 553, 796, 3770
exp().....	798
exp2().....	800
exp2f().....	800
exp2l().....	800
expand	58, 2770, 3772
expf().....	798
expl().....	798
expm1().....	802
expm1f().....	802
expm1l().....	802
export.....	2401
expr	2773 , 3771-3772
matching expression.....	2775
expression argument	2495
expression list	2495
EXPR_NEST_MAX	275, 2099, 2334, 3710
EXTA	477
EXTB.....	477
extended regular expression.....	58, 188, 2491, 2609, 2800, 2842, 2888, 3019, 3078, 3199 3449, 3545
extended security controls	58, 108, 3511
extension	
CX.....	7
OH.....	9
XSI	12
F-LOCK.....	444
fabs().....	804
fabsf().....	804
fabsl().....	804
faccessat().....	570, 806
false	2344, 2367, 2778, 3717, 3771
fattach().....	807
fc.....	2344, 2367, 2780, 3717, 3772
fchdir().....	810
fchmod().....	811 , 3768
fchmodat()	665, 813
fchown().....	814
fchownat().....	670, 816
fclose().....	817 , 3768
fcntl()	820 , 3548, 3572, 3602, 3768
fcntl() locks	3661
fcvt().....	3692
FD	7
fdatasync().....	829
fdetach().....	831
fdim().....	833

fdimf()	833
fdiml()	833
fdopen()	835, 3601
fdopendir()	838
fds_	475
FD_	475
fd_	475
FD_CLOEXEC	238, 497, 649, 753, 784, 820, 840, 1127, 1408, 1453, 1460, 1933
FD_CLR	1553
FD_CLR()	552
FD_ISSET	552, 1553
fd_set	380, 399
FD_SET	552, 1553
FD_SETSIZE	380
FD_ZERO	552, 1553
feature test macro	59, 472, 1014, 3567, 3689
_POSIX_C_SOURCE	472
_XOPEN_SOURCE	473
feclearexcept()	842
fegetenv()	843
fegetexcept ag()	844
fegetround()	845
feholdexcept()	847
fenv_t	243
feof()	848
feraiseexcept()	849
ferror()	850
fesetenv()	843, 851
fesetexcept ag()	844, 852
fesetround()	845, 853
fetestexcept()	854
feupdateenv()	856
fexcept_t	243
fexecve	858
fexecve()	783
FE_	477
FE_constants	
defined in <fenv.h>	243
FE_ALL_EXCEPT	243
FE_DFL_ENV	244
FE_DIVBYZERO	243
FE_DOWNWARD	243
FE_INEXACT	243
FE_INVALID	243
FE_OVERFLOW	243
FE_TONEAREST	243
FE_TOWARDZERO	243
FE_UNDERFLOW	243
FE_UPWARD	243
FFDLY	417
fush()	859
FFn	417

ffs()	862
fg	2344, 2367, 2786, 3717, 3772
fgetc()	863, 3769
fgetpos()	865
fgets()	867
fgetwc()	869
fgetws()	871
eld	59
field splitting	2359, 3734
FIFO	59, 1322, 1324, 1414, 2314, 3492, 3500, 3611
FIFO special file	59, 3002, 3492
FIFOTYPE	413
file	59
FILE	356, 458, 547
le	2788, 3492
locking	826
file access permissions	108, 2328, 3511
file accessibility	572
file characteristics	
data structure	395
header	395
file classes	3492
file comparisons	
cmp	2588
comm	2592
diff	2658
uniq	3346
file contents	2330
file control	826
le conversion	
cut	2626
dd	2641
expand	2770
fold	2806
head	2850
join	2874
od	3044
paste	3052
patch	3056
sort	3247
strings	3259
tail	3277
tr	3310
tsort	3319
unexpand	3340
uniq	3346
uudecode	3357
uuencode	3360
le creation	2328, 3708
file description	59
file descriptor	60, 2327, 2360, 2368, 3585, 3691, 3735
file format notation	3520

file group class	60
file hierarchy	109, 3512
file hierarchy manipulation	3766, 3773
file mode	60
file mode bits.....	60
file mode creation mask	2327
FILE object.....	496
file offset	61
file other class	61
file owner class	61
file permission bits	61, 572
file permission commands	
chgrp	2569
chmod	2572
chown.....	2579
umask.....	3328
file permissions	572, 905, 968, 3511, 3551
file position indicator.....	495
file read	2328, 3708
file removal.....	2330, 3708
file searching	
grep.....	2842
file serial number.....	61
file size, arbitrary.....	3716
file system.....	61, 3492
file system, mounted.....	3498
file system, root.....	3501
file time values.....	2330
file times update	109, 3514
file tree commands	
diff	2658
nd	2796
ls.....	2923
mkdir.....	2999
rmdir	3205
file type	62
file write.....	2328, 3708
file, passwd.....	3500
filename	60, 109, 3492, 3512
filename portability.....	109, 3514
filename string.....	61
FILENAME_MAX.....	355
fileno()	873, 3499
FILESIZEBITS	273, 902
lter	62
filtering trace event types.....	3685
filters	
asa	2470
awk	2482
compress.....	2602
dd.....	2641
expand	2770

fold	2806
head	2850
iconv	2853
more	3005
nl	3031
paste	3052
pax	3068
pr	3107
read	3191
sed	3216
tail	3277
tee	3285
tr	3310
uncompress	3337
unexpand	3340
zcat	3471
FIND	1117
nd	2796 , 3772-3773
find string token	2079
FIPS	17
FIPS requirements	3479
first open (of a file)	62
flockfile()	874 , 3575
oor()	876
oorf()	876
oorl()	876
ow control	62
FLT_ constants	
defined in <float.h>	248
FLT_DIG	249
FLT_EPSILON	250
FLT_EVAL_METHOD	247
FLT_MANT_DIG	248
FLT_MAX	250
FLT_MAX_10_EXP	250
FLT_MAX_EXP	249
FLT_MIN	250, 603, 605, 609, 611, 615, 776, 798, 800, 802, 833, 885
FLT_MIN_10_EXP	249
FLT_MIN_EXP	249
FLT_RADIX	248, 1280
FLT_ROUNDS	247, 878
FLUSH	475
FLUSHO	477
FLUSHR	368, 1147
FLUSHRW	368, 1147
FLUSHW	368, 1147
fma()	878
fmaf()	878
fmal()	878
fmax()	880
fmaxf()	880
fmaxl()	880

fmemopen()	881
fmin()	884
fminf()	884
fminl()	884
FMNAMESZ	367-368, 1146
fmod()	885
fmodf()	885
fmodl()	885
fmsg()	887
fnmatch()	890, 3774
FNM_	475
FNM_ constants	
in <fnmatch.h>	254
FNM_NOESCAPE	254, 890
FNM_NOMATCH	254, 890
FNM_PATHNAME	254, 890
FNM_PERIOD	254, 890
fold	2806, 3772
fopen()	892, 3493, 3716, 3768
FOPEN_MAX	272, 355, 835, 882, 894, 2162
for loop	2372, 3743
foreground	1909, 3494-3497, 3549-3551
foreground job	62
foreground process	62
group	62
group ID	62
fork()	897, 3494, 3550, 3591, 3599, 3601, 3682, 3689, 3694, 3768
forkall	900
form-feed character	63
format of entries	219, 469
fort77	2810, 3774
external symbols	2813
standard libraries	2812
fpathconf()	902, 3767, 3769
fpclassify()	908
FPE_	475
FPE_FLTDIV	337
FPE_FLTINV	337
FPE_FLTOVF	337
FPE_FLTRES	337
FPE_FLTSUB	337
FPE_FLTUND	337
FPE_INTDIV	337
FPE_INTOVF	337
fprintf()	909
fputc()	923, 3769
fputs()	925
fputwc()	927
fputws()	929
FP_ILOGB0	1133
FP_ILOGBNAN	1133
FQDN	1057

FR.....	7
frac_digits.....	156
fread().....	931, 3769
free().....	933, 3583, 3658
freeaddrinfo().....	934
freelocale().....	940
freopen().....	942
frexp().....	947
frexpf().....	947
frexpl().....	947
fsblkcnt_t.....	402
FSC.....	8
fscanf().....	949
fseek().....	957, 3769
fseeko().....	957
fsetpos().....	960, 3769
fs_lcnt_t.....	402
fstat().....	962, 3768
fstatat().....	965
fstatvfs().....	971
fsync().....	974, 3595
ftell().....	976
ftello(0.....	976
ftime().....	3692
ftok().....	978
ftruncate().....	980, 3601, 3603, 3768
ftrylock le().....	874, 982
FTW.....	255, 475, 1397-1398
ftw().....	983, 3716
FTW_constants	
in <ftw.h>.....	255
FTW_CHDIR.....	255, 1397
FTW_D.....	255, 983, 1397
FTW_DEPTH.....	255, 1397
FTW_DNR.....	255, 983, 1397-1398
FTW_DP.....	255, 1397
FTW_F.....	255, 983, 1397
FTW_MOUNT.....	255, 1397
FTW_NS.....	255, 983, 1397-1398
FTW_PHYS.....	255, 1397
FTW_SL.....	255, 983, 1397
FTW_SLN.....	255, 1398
fully-qualified domain name.....	1057
function definition command.....	2374, 3744
function identifiers.....	2529
functions.....	471
implementation.....	471, 3565
use.....	471, 3565
funlock le().....	874, 986
fuser.....	2816
futimens().....	987
fwide().....	991

fwprintf()	993
fwrite()	1001, 3769
fwscanf()	1003
f_	476
F_	477
F_DUPFD	238, 820, 823
F_DUPFD_CLOEXEC	238, 820, 823
F_GETFD	238, 820, 823
F_GETFL	238, 820, 823
F_GETLK	238, 821, 823
F_GETOWN	238, 821, 823
F_LOCK	518, 1269
F_OK	441
F_RDLCK	238, 823
F_SETFD	238, 820, 823
F_SETFL	238, 820, 823
F_SETLK	238, 821, 823
F_SETLKW	238, 518, 821, 823
F_SETOWN	238, 821, 823
F_TEST	444, 1269
F_TLOCK	444, 1269
F_ULOCK	444, 1269
F_UNLCK	238, 821-822
F_WRLCK	238
g-file	2650
gai_strerror()	1010
gcvt()	3692
gencat	2819, 3773
escape sequences	2820
general terminal interface	3548
generated file	2650
generation-version attribute	542, 1492
get	2823
get configurable pathname variables	904
get configurable system variables	2102
get file status	968
get process times	2159
get supplementary group IDs	1042
get system time	2149
get thread ID	1726
get user name	1051
getaddrinfo()	934, 1011
GETALL	382, 1867
getc()	1012, 3648, 3769
getch()	3769
getchar()	1015
getchar_unlocked()	1013, 1016
getconf	2831, 3709, 3767, 3774
getcontext()	3692
getcwd()	1017
getc_unlocked()	1013
getdate()	1020

getdate_err	1020
getdelim().....	1025
getegid().....	1028 , 3768
getenv().....	792, 1029
geteuid().....	1032 , 3768
getgid().....	1033 , 3768
getgrent().....	758, 1034, 3510
getgrgid().....	1035 , 3510, 3649, 3768
getgrgid_r().....	1035
getgrnam().....	1039 , 3510, 3514, 3649, 3768
getgrnam_r().....	1039
getgroups().....	1042 , 3502
gethostbyaddr().....	3692
gethostbyname().....	3692
gethostent().....	760, 1044
gethostid().....	1045
gethostname().....	1046
getitimer().....	1047
getline().....	1025, 1049
getlogin().....	1050 , 3768
getlogin_r().....	1050
getmsg().....	1053
getnameinfo().....	1057
GETNCNT.....	382, 1867-1868
getnetbyaddr().....	762, 1060
getnetbyname().....	762, 1060
getnetent().....	762, 1060
getopt().....	1061 , 3556, 3773-3774
getopts.....	2344, 2367, 2837, 3717, 3774
getpeername().....	1066
getpgid().....	1068
getpgrp().....	1069 , 3496
GETPID.....	382, 1867-1868
getpid().....	1070 , 3585, 3768
getpmsg().....	1053, 1071
getppid().....	1072 , 3768
getpriority().....	1073 , 3612
getprotent().....	1076
getprotobyname().....	764, 1076
getprotobynumber().....	764, 1076
getprotoent().....	764
getpwent().....	766, 1077, 3510
getpwnam().....	1078 , 3510, 3514, 3649, 3768
getpwnam_r().....	1078
getpwuid().....	1082 , 3510, 3649, 3768
getpwuid_r().....	1082
getrlimit().....	1086 , 3716
getrusage().....	1089 , 3625
gets().....	1091
getservbyname().....	769, 1093
getservbyport().....	769, 1093
getservent().....	769, 1093

getsid().....	1094
getsockname()	1095
getsockopt().....	1097
getsubopt().....	1099
gettimeofday().....	1103
getty.....	3550
getuid().....	1104, 3585, 3690, 3768
getutxent().....	771, 1106
getutxid().....	771, 1106
getutxline().....	771, 1106
GETVAL	382, 1867-1868
getwc().....	1107
getwchar().....	1108
getwd().....	3692
GETZCNT	382, 1867-1868
gid_t	402, 3510
glob().....	1109, 3774
global storage class	2533
globfree().....	1109, 3774
GLOB_	475
GLOB_ constants	
defined in <glob.h>.....	257
error returns of glob	1111
used in glob	1109
GLOB_ABORTED.....	257, 1111
GLOB_APPEND.....	257, 1109-1110
GLOB_DOOFFS.....	257, 1109-1110
GLOB_ERR.....	257, 1109, 1111
GLOB_MARK	257, 1109
GLOB_NOCHECK	257, 1110-1111
GLOB_NOESCAPE	257, 1110
GLOB_NOMATCH	257, 1111
GLOB_NOSORT	257, 1110
GLOB_NOSPACE	257, 1111
gl_	475
GMT0	2185
gmtime().....	1113, 3519
gmtime_r()	1113
GNU make	2987
grammar	
conventions	3712
locale	166
regular expression.....	192
grantpt()	1115
graphic character.....	63
grep	2842, 3772-3773
group database	63, 3493
group database access	3510
group file	3493
group ID	63
group name	63
grouping commands.....	2371, 3743

HALT.....	888
hard limit.....	64
hard link.....	64
hash.....	2344, 2367, 2847
hcreate().....	1117
hdestroy().....	1117
head.....	2850 , 3772
headers.....	219 , 3558
here-document.....	2362, 3736
high resolution sleep.....	1389
historical implementations.....	3493
history command	
fc.....	2780
hold batch job request.....	2443
Hold_Types.....	2437
HOME.....	177 , 2714, 3479
home directory.....	64
host byte order.....	64, 110, 3514
host name.....	934
hosted implementation.....	3493
hostent.....	302
HOST_NAME_MAX.....	271
hsearch().....	1117
htonl().....	1120
htons().....	1120
HUGE_VAL.....	289, 2271
HUGE_VALF.....	289
HUGE_VALL.....	289
hunk.....	3058
HUPCL.....	418
hypot().....	1121
hypotf().....	1121
hypotl().....	1121
h.....	475
h_errno.....	3693
I.....	224
ICANON.....	418, 3552, 3554
iconv.....	2853
iconv().....	1123 , 3773
iconv_close().....	1126 , 3773
iconv_open().....	1127 , 3773
ICRNL.....	416
ic.....	475
id.....	2857 , 3774
idtype_t.....	409
id_t.....	402
IEEE Std 754-1985.....	469
IEEE Std 854-1987.....	469
IEXTEN.....	418
if.....	3744
if conditional construct.....	2373
ifc.....	476

ifra_.....	476
ifru_.....	476
IF_.....	475
if_.....	475-476
if_freenameindex().....	1129
if_indextoname().....	1130
if_nameindex.....	301
if_nameindex().....	1131
IF_NAMESIZE.....	301
if_nametoindex().....	1132
IGNBRK.....	416
IGNCR.....	416
IGNPAR.....	416
ILL_.....	475
ILL_BADSTK.....	337
ILL_COPROC.....	337
ILL_ILLADR.....	337
ILL_ILLOPC.....	337
ILL_ILLOPN.....	337
ILL_ILLTRP.....	337
ILL_PRVOPC.....	337
ILL_PRVREG.....	337
ilogb().....	1133
ilogbf().....	1133
ilogbl().....	1133
imaginary.....	224
imaxabs().....	1135
imaxdiv().....	1136
implementation.....	3493
historical.....	3493
hosted.....	3493
native.....	3498
specific.....	3493
implementation-defined.....	53480-3481
rationale.....	3480
IMPLINK_.....	477
in6_.....	475
IN6_.....	477
IN6_IS_ADDR_LINKLOCAL.....	308
IN6_IS_ADDR_LOOPBACK.....	308
IN6_IS_ADDR_MC_GLOBAL.....	309
IN6_IS_ADDR_MC_LINKLOCAL.....	309
IN6_IS_ADDR_MC_NODELOCAL.....	308
IN6_IS_ADDR_MC_ORGLOCAL.....	309
IN6_IS_ADDR_MC_SITELOCAL.....	309
IN6_IS_ADDR_MULTICAST.....	308
IN6_IS_ADDR_SITELOCAL.....	308
IN6_IS_ADDR_UNSPECIFIED.....	308
IN6_IS_ADDR_V4COMPAT.....	308
IN6_IS_ADDR_V4MAPPED.....	308
INADDR_.....	475
include line.....	2973

incomplete line	64
incomplete pathname	3494
index()	3693
INET6_ADDRSTRLEN.....	308
inet_.....	475
inet_addr()	1137
INET_ADDRSTRLEN.....	308
inet_ntoa().....	1137
inet_ntop()	1139
inet_pton().....	1139
Inf.....	64, 603, 605, 609, 615
INF.....	912, 996
inference rule	2969
INFINITY.....	289, 912, 996
INFO.....	888
infu_.....	476
inheritance attribute.....	542, 1494
init.....	556, 1227
initialization	3511
initialize a named semaphore.....	1855
initializing a mutex	1662
initstate()	1141
INIT_PROCESS	456, 771-772
INLCR.....	416
ino_t.....	402
INPCK.....	416
input and output rationale.....	1774
input file descriptor	
duplication	3736
input mode.....	2677, 3553
input processing	3551
canonical mode.....	3551
non-canonical mode.....	3552
insque().....	1143
instrumented application.....	64
INT	477
inter-user communication.....	3766, 3773
interactive facilities	3765, 3772
interactive shell.....	65
interface	3662
characteristics.....	3549
international environment	1903
internationalization.....	65
internationalization variable	3537
Internet Protocols	532
interprocess communication	65, 3586
INTMAX_MAX	352
INTMAX_MIN	352
INTN_MAX.....	351
INTN_MIN.....	351
INTPTR_MAX	352
INTPTR_MIN	351

int_curr_symbol	155
INT_FASTN_MAX	351
INT_FASTN_MIN	351
int_frac_digits	156
INT_LEASTN_MAX	351
INT_LEASTN_MIN	351
INT_MAX	280, 1133
INT_MIN	280, 567
int_n_cs_precedes	156
int_n_sep_by_space	157
int_n_sign_posn	157
int_p_cs_precedes	156
int_p_sep_by_space	157
int_p_sign_posn	157
invalid	182
use in RE	3541
invariant values	281
invoke	65
in_	475
IN_	477
in_addr	306
ioctl()	1146 , 3548, 3573
iovec	406
iov_	476
IOV_	477
IOV_MAX	271, 406, 1786, 2099, 2319
IP6	8
IPC	371, 501, 1372, 1374, 1377, 1379, 1871, 1876, 1944, 1947, 3586
ipcrm	2861
ipcs	2863
IPC_	475
ipc_	475
IPC_constants	
defined in <sys/ipc.h>	371
used in semctl	1868
used in shmctl	1942
IPC_CREAT	371, 1373, 1870, 1946
IPC_EXCL	371, 1373, 1870
IPC_NOWAIT	371, 1375-1376, 1378-1379, 1872
IPC_PRIVATE	371, 1373, 1870, 1946
IPC_RMID	371, 1371, 1868, 1942
IPC_SET	371, 1371, 1868, 1942
IPC_STAT	371, 1371, 1868, 1942
IPPORT_	477
IPPROTO_	475
IPPROTO_ICMP	307
IPPROTO_IP	307
IPPROTO_IPV6	307
IPPROTO_RAW	307
IPPROTO_TCP	307
IPPROTO_UDP	307
IPv4	532

IPv4-compatible address.....	533
IPv4-mapped address.....	533
IPv6.....	532
compatibility with IPv4.....	533
interface identification.....	534
options	534
IPv6 address	
anycast	533
loopback	533
multicast	533
unicast.....	532
unspecified	533
IPV6_.....	475
IPV6_JOIN_GROUP	308, 534
IPV6_LEAVE_GROUP.....	308, 534
ipv6_mreq.....	307
IPV6_MULTICAST_HOPS.....	308, 534
IPV6_MULTICAST_IF.....	308, 534
IPV6_MULTICAST_LOOP	308, 534
IPV6_UNICAST_HOPS.....	308, 535
IPV6_V6ONLY.....	308, 535
ip_.....	475
IP_.....	477
isalnum().....	1158
isalnum_l().....	1158
isalpha().....	1160
isalpha_l().....	1160
isascii().....	1162
isastream().....	1163
isatty().....	1164
isblank().....	1165
isblank_l().....	1165
iscntrl()	1167
iscntrl_l()	1167
isdigit()	1169
isdigit_l()	1169
is_nite().....	1171
isgraph()	1172
isgraph_l().....	1172
isgreater()	1174
isgreaterequal()	1175
ISIG.....	418
isinf().....	1176
isless()	1177
islessequal()	1178
islessgreater().....	1179
islower()	1180
islower_l()	1180
isnan().....	1183
isnormal().....	1184
ISO/IEC 646: 1991 standard.....	3500
ISO C standard	224, 469, 595, 791, 826, 1014, 1292, 1768, 1819, 1903, 1955, 1971, 1982

.....	2149, 3487, 3489, 3519, 3548, 3565
isprint()	1185
isprint_l()	1185
ispunct()	1187
ispunct_l()	1187
isspace()	1189
isspace_l()	1189
ISTRIP	416, 3553
isunordered()	1191
isupper()	1192
isupper_l()	1192
iswalnum()	1194
iswalnum_l()	1194
iswalpha()	1196
iswalpha_l()	1196
iswblank()	1198
iswblank_l()	1198
iswcntrl()	1200
iswcntrl_l()	1200
iswctype()	1202
iswctype_l()	1202
iswdigit()	1205
iswdigit_l()	1205
iswgraph()	1207
iswgraph_l()	1207
iswlower()	1209
iswlower_l()	1209
iswprint()	1211
iswprint_l()	1211
iswpunct()	1213
iswpunct_l()	1213
iswspace()	1215
iswspace_l()	1215
iswupper()	1217
iswupper_l()	1217
iswxdigit()	1219
iswxdigit_l()	1219
isxdigit()	1221
isxdigit_l()	1221
itimerspec	425
itimerval	399
ITIMER_	476
ITIMER_PROF	399, 1047
ITIMER_REAL	399, 1047
ITIMER_VIRTUAL	399, 1047
it_	476
IXANY	416
IXOFF	416
IXON	416
I_	475
I_ATMARK	367, 1153
I_CANPUT	367, 1153

I_CKBAND.....	367, 1153
I_FDINSERT.....	367, 1149
I_FIND.....	367, 1148
I_FLUSH.....	367, 1146
I_FLUSHBAND.....	367, 1147
I_GETBAND.....	367, 1153
I_GETCLTIME.....	367, 1154
I_GETSIG.....	367, 1148
I_GRDOPT.....	367, 1149, 1772
I_GWROPT.....	367, 1151
I_ISVTX.....	2574
I_LINK.....	367, 1154
I_LIST.....	367, 1152
I_LOOK.....	367, 1146
I_NREAD.....	367, 1149
I_PEEK.....	367, 1148
I_PLINK.....	367, 1155
I_POP.....	367, 1146
I_PUNLINK.....	367, 1155
I_PUSH.....	367, 1146
I_RECVFD.....	367, 482, 1152
I_SENDFD.....	367, 1151-1152
I_SETCLTIME.....	367, 688, 1153
I_SETSIG.....	367, 1147-1148
I_SRDOPT.....	367, 1149, 1772
I_STR.....	367, 1150
I_SWROPT.....	368, 1151, 2312
I_UNLINK.....	368, 1154
j0().....	1223
j1().....	1223
jn().....	1223
job.....	65
job control.....	65, 556, 1069, 1227, 1909, 1922, 2102, 2232, 3494-3497, 3499, 3549, 3551
.....	3576, 3582, 3768
implementing applications.....	3496
implementing shells.....	3494
implementing systems.....	3497
job control job ID.....	66
jobs.....	2344, 2367, 2870, 3717, 3772
Job_Owner.....	2437
join.....	2874 , 3772
Join_Path.....	2437
jrand48().....	749, 1225
JST-9.....	2185
Keep_Files.....	2437
kernel.....	3497
kernel entity.....	3651
keyword-value pairs.....	2451
key_t.....	402
kill.....	2344, 2367, 2879, 3717, 3772
kill().....	1226 , 3576-3577, 3580, 3582-3583, 3689
killpg().....	1229

l64a()	563, 1231
labs()	1232
LANG	174 , 649
last close	1938, 3602
last close (of a file)	66
LASTMARK	369, 1153
lchown()	1233
lcong48()	749, 1236
LC_ALL	175 , 285, 785, 1262, 1403, 1902, 1904
LC_COLLATE	175 , 275, 285, 1109-1110, 1902, 1904, 2027, 2090, 2249, 2290, 3528
description	147
LC_CTYPE	175 , 267, 285, 463, 1202, 1297, 1306, 1308, 1902, 1904, 2167, 2173, 3527
description	139
LC_GLOBAL_LOCALE	754, 940
LC_MESSAGES	175 , 267, 285, 312, 649, 1902-1904, 2035, 3532
description	165
LC_MONETARY	175 , 267, 285, 1262, 1902, 1904, 2041, 3530
description	155
LC_NUMERIC	175 , 267, 285, 910, 949, 993, 1003, 1262, 1902, 1904, 2041, 2073, 2271
description	3531, 3557
description	158
LC_TIME	175 , 267, 285, 1021, 1404, 1902, 1904, 3531
description	159
ld	
rationale for omission	3757
LDBL_constants	
defined in <float.h>	248
LDBL_DIG	249
LDBL_EPSILON	250
LDBL_MANT_DIG	248
LDBL_MAX	250
LDBL_MAX_10_EXP	250
LDBL_MAX_EXP	249
LDBL_MIN	250, 603, 605, 609, 611, 615, 776, 798, 800, 802, 833, 885
LDBL_MIN_10_EXP	249
LDBL_MIN_EXP	249
ldexp()	1237
ldexpf()	1237
ldexpl()	1237
ldiv()	1239
legacy	5, 3481
rationale	3481
lex	2884 , 3774, 3776
actions	2890
definitions	2887
escape sequences	2889
regular expressions	2888
rules	2888
table sizes	2887
translation table	2895
user subroutines	2888
l nd()	1240 , 1290

lgamma()	1241
lgammaf()	1241
lgammal()	1241
libraries	
ar command	2462
library routine	3497
LIMIT	2333
limit	
numerical	280
limits	3709
line	66
rationale for omission	3757
line counting	3434
LINES	177, 3538
LINE_MAX	275, 2099, 2334, 2483, 2690, 2698, 2965, 3224, 3349, 3498, 3509, 3710
linger	66
link	66, 2896
link count	66
link to a file	1246
link()	1243, 3491, 3769
linkat()	1243
LINK_MAX	273, 484, 902, 1244, 1817, 3710, 3783
lint	
rationale for omission	3757
LIO_	475
lio_	475
lio_listio()	1248, 3580, 3596
LIO_NOP	220, 1248
LIO_NOWAIT	220, 1248
LIO_READ	220, 1248
LIO_WAIT	220, 1248
LIO_WRITE	220, 1248
list directed I/O	1250
listen()	1252
lists	2369, 3741
AND-OR	2369
compound-list	2369
live process	66
llabs()	1232, 1254
lldiv()	1239, 1255
LLONG_MAX	280, 2082, 2278
LLONG_MIN	280, 2082, 2278
llrint()	1256
llrintf()	1256
llrintl()	1256
llround()	1258
llroundf()	1258
llroundl()	1258
ln	2898, 3716, 3773
LNKTYPE	413
load order	744
LOBLK	477

local customs.....	67
local IPC.....	67
local mode.....	3554
locale.....	67, 135, 2903, 3525, 3773
con guration.....	3766,3773
de nition.....	136,3526
definition example.....	3533
definition grammar.....	3533
grammar.....	166, 3533
lexical conventions.....	3533
POSIX.....	136
localeconv().....	1260
localedef.....	2909, 3773-3774
localization.....	67
localtime().....	1265, 3519, 3647
localtime_r().....	1265
locate batch job request.....	2444
lockf().....	1269
locking.....	826
advisory.....	826
mandatory.....	826
locking and unlocking a mutex.....	1673
locking file.....	2576
log().....	1272
log-full-policy attribute.....	540, 542, 1494, 1497, 1514
log-max-size attribute.....	542, 1495, 1497
log10().....	1274
log10f().....	1274
log10l().....	1274
log1p().....	1276
log1pf().....	1276
log1pl().....	1276
log2().....	1278
log2f().....	1278
log2l().....	1278
logb().....	1280
logbf().....	1280
logbl().....	1280
logf().....	1272, 1282
logger.....	2913, 3774
logical device.....	3497
login.....	67
rationale for omission.....	3758
login name.....	67
login shell.....	791
LOGIN_NAME_MAX.....	271, 1050, 2099, 3783
LOGIN_PROCESS.....	456, 771-772
logl().....	1272, 1282
LOGNAME.....	178
logname.....	2916
LOGNAME.....	3479, 3538
logname.....	3774

LOG_.....	476
LOG_ constants in syslog.....	694
LOG_ALERT.....	412, 694
LOG_AUTH.....	411
LOG_CONS.....	411, 695
LOG_CRIT.....	412, 694
LOG_CRON.....	411
LOG_DAEMON.....	411
LOG_DEBUG.....	412, 694
LOG_EMERG.....	412, 694
LOG_ERR.....	412, 694
LOG_INFO.....	412, 694
LOG_KERN.....	411
LOG_LOCAL.....	411, 694
LOG_LPR.....	411
LOG_MAIL.....	411
LOG_MASK.....	411
LOG_NDELAY.....	411, 695
LOG_NEWS.....	411
LOG_NOTICE.....	412, 694
LOG_NOWAIT.....	411, 695
LOG_ODELAY.....	411, 695
LOG_PID.....	411, 695
LOG_USER.....	411, 694-695
LOG_UUCP.....	411
LOG_WARNING.....	412, 694
longjmp().....	1283 , 3572, 3583, 3655, 3657, 3771
LONG_BIT.....	280
LONG_MAX.....	280, 2082, 2278, 3555
LONG_MIN.....	280, 2082, 2278, 3555
lorder	
rationale for omission.....	3758
lower multiplexing.....	94
lp.....	2918 , 3774
lpstat	
rationale for omission.....	3758
LR(1) grammars.....	3468
lrand48().....	749, 1285
lrint().....	1286
lrintf().....	1286
lrintl().....	1286
lround().....	1288
lroundf().....	1288
lroundl().....	1288
ls.....	2923 , 3716, 3773
lsearch().....	1290
lseek().....	1292 , 3595-3596, 3601, 3646, 3769
lstat().....	965, 1294, 3716, 3768
l.....	475-476
L_ANCHOR.....	192
L_ctermid.....	355, 725
l_sysid.....	826

L_tmpnam	355
m4	2933
macro	3482
macro processor	2933
MAGIC	227
magic file	2793
mail	
rationale for omission	3758
mailx	2943, 3772-3773
change current directory	2954
change folder	2956
command escapes	2962
commands	2953
copy messages	2954
declare aliases	2954
declare alternatives	2954
delete aliases	2960
delete messages	2955
delete messages and display	2955
direct messages to mbox	2958
discard header fields	2955
display beginning of messages	2960
display current message number	2962
display header summary	2957
display list of folders	2956
display message	2958
display message size	2960
echo a string	2955
edit message	2955, 2961
execute commands conditionally	2957
exit	2956
follow up specified messages	2956
help	2957
hold messages	2957
internal variables	2950
invoke a shell	2960
invoke shell command	2961
list available commands	2957
mail a message	2958
null command	2962
pipe message	2958
process next specified message	2958
quit	2959
read mailx commands from a file	2960
receive mode	2943
reply to a message	2959
reply to a message list	2959
retain header fields	2959
save messages	2959
scroll header display	2961
send mode	2943
set variables	2960

start-up.....	2950
touch messages.....	2960
undelete messages.....	2961
unset variables.....	2961
write messages to a file.....	2961
Mail_Points	2438
Mail_Users	2438
main()	3583
make.....	2969 , 3774-3775
default rules	2980
inference rules.....	2977
internal macros	2979
libraries	2978
macros.....	2976
makefile execution	2974
makefile syntax.....	2972
target rules.....	2974
make, GNU version	2987
makecontext()	3692-3693
malloc()	1295 , 3583, 3605-3606, 3634, 3647-3648, 3658-3659
man.....	2992 , 3772
manipulate signal sets	1961
map.....	67, 3497
mapped.....	3497
mappings.....	1343
MAP_	475
MAP_FAILED	1344
MAP_FIXED	373, 1339
MAP_PRIVATE.....	373, 897, 1339, 1343, 1348, 1381
MAP_SHARED	373, 900, 1339-1340
margin code.....	3482
notation.....	13, 3483
marked message.....	68
matched	68, 181
mathematical functions	2333
domain error	117
error conditions	117, 3520
NaN arguments	118, 3520
pole error	117
range error	118
max-data-size attribute.....	542, 1497
MAXARGS	344
MAXFLOAT	289
maximum values.....	275
MAX_CANON	273, 902, 3552, 3710, 3783
MAX_INPUT	273, 902, 3710, 3783
may.....	5, 3480
rationale.....	3480
mblen()	1297
mbrlen().....	1299
mbrtowc()	1301
mbsinit()	1303

mbsnrtowcs()	1304
mbsrtowcs()	1304
mbstowcs()	1306
mbtowc()	1308
MB_CUR_MAX	359, 1297, 1299, 1301, 1308, 2242, 2292
MB_LEN_MAX	280
MC1	8
MCL_	475
MCL_CURRENT	373, 1336
MCL_FUTURE	373, 1336, 1344, 3599
MCL_INHERIT	3600
mcontext_t	335
memccpy()	1310
memchr()	1311
memcmp()	1312
memcpy()	1313
memmove()	1314
memory locking	3597
memory management	505, 3597, 3768
memory management unit	3598
memory mapped files	68
memory object	68, 3498
memory synchronization	111, 3515
memory-resident	68, 3497
memset()	1315
mesg	2996 , 3773-3774
message	69
message catalog	69
descriptor	69, 553, 785, 790
generation	2819
message parts	501
message passing	3589, 3768, 3770
message priority	500
high-priority	500
normal	500
priority	500
message queue	69, 3589
message-discard mode	1772
message-nondiscard mode	1772
MET-1MEST	2185
META_CHAR	192
MIL-STD-1753	2814
minimum values	276
Minimum_Cpu_Interval	2435
MINSIGSTKSZ	335, 1958
mkdir	2999 , 3773
mkdir()	1316 , 3769
mkdirat()	1316
mkdtemp()	1319
mk fo	3002 , 3773
mk fo()	1322 , 3493
mk foat()	1322

mknod	
rationale for omission.....	3758
mknod().....	1326 , 3493
mknodat().....	1326
mkstemp().....	1319, 1330
mktemp().....	3693
mktime().....	1331 , 3519
ML.....	8
mlock().....	1334
mlockall().....	1336 , 3599
MLR.....	8
mmap().....	1338 , 3601-3605
MMU.....	3598
MM_.....	475
MM_ macros.....	252
MM_APPL.....	252, 887
MM_CONSOLE.....	252, 887
MM_ERROR.....	252, 888-889
MM_FIRM.....	252
mm_FIRM.....	887
MM_HALT.....	252, 888
MM_HARD.....	252, 887
MM_INFO.....	252, 888
MM_NOCON.....	253, 888
MM_NOMSG.....	252, 888
MM_NOSEV.....	252, 888
MM_NOTOK.....	252, 888
MM_NRECOV.....	252, 887
MM_NULLACT.....	252
MM_NULLLBL.....	252
MM_NULLMC.....	252, 887
MM_NULLSEV.....	252
MM_NULLTAG.....	252
MM_NULLTXT.....	252
MM_OK.....	252, 888
MM_OPSYS.....	252, 887
MM_PRINT.....	252, 887, 889
MM_RECOVER.....	252, 887
MM_SOFT.....	252, 887
MM_UTIL.....	252, 887
MM_WARNING.....	252, 888
mode.....	69
modem disconnect.....	3552
mode_t.....	402
modf().....	1346
modff().....	1346
mod ().....	1346
modify batch job request.....	2444
MON.....	8
monotonic clock.....	69, 3621
MON_.....	267
mon_decimal_point.....	155

mon_grouping	155
mon_thousands_sep	155
more	3005 , 3772-3773
discard and refresh	3011
display position	3014
examine new file	3013
examine next file	3013
examine previous file	3013
go to beginning of file	3011
go to end-of-file	3011
go to tag	3013
help	3010
invoke editor	3013
mark position	3011
quit	3014
refresh the screen	3011
repeat search	3012
repeat search in reverse	3012
return to mark	3012
return to previous position	3012
scroll backward one half screenful	3011
scroll backward one line	3010
scroll backward one screenful	3010
scroll forward one half screenful	3010
scroll forward one line	3010
scroll forward one screenful	3010
search backward for pattern	3012
search forward for pattern	3012
skip forward one line	3011
MORECTL	369, 1054
MOREDATA	369, 1054
motion command	3232
mount point	70, 3498
mount()	3498
mounted file system	3498
move batch job request	2445
mprotect()	1348 , 3601
MQ	475
mq	475
mq_close()	1350
mq_getattr()	1351
mq_notify()	1353
mq_open()	1356 , 3590
MQ_OPEN_MAX	271, 2099, 3783
MQ_PRIO_MAX	271, 1362-1363, 2099, 3783
mq_receive()	1359 , 3590
mq_send()	1362 , 3590
mq_setattr()	1364
mq_timedreceive()	1359, 1366, 3622
mq_timedsend()	1362, 1367, 3622
mq_unlink()	1368
mrnd48()	749, 1370

MSG	8
msg	475
msg*()	3586
msgctl()	1371, 3587
msgget()	1373, 3587
msgrcv()	1375, 3587
msgsnd()	1378, 3587
MSGVERB	178, 888-889
MSG_	475-476
msg_	476
MSG_ANY	369, 1053
MSG_BAND	369, 1053, 1753
MSG_CTRUNC	388
MSG_DONTROUTE	388
MSG_EOR	388, 1877, 1880, 1884, 2004, 2006
MSG_HIPRI	369, 1053, 1753
MSG_NOERROR	376, 1375-1376
MSG_NOSIGNAL	388, 1877, 1880, 1884
MSG_OOB	388, 1877, 1880, 1884
MSG_PEEK	388
msg_perm	502
MSG_TRUNC	388
MSG_WAITALL	388
msgid	502
MST7MDT	2185
msync()	1381, 3601
MS_	475
MS_ASYNC	373, 1340, 1381
MS_INVALIDATE	373, 1381-1382
MS_SYNC	373, 1340, 1381
multi-byte character	3551, 3553
multi-character collating element	70
multi-threaded library	70
multi-threaded process	70
multi-threaded program	70
multicast	533
multiple tasks	3765, 3772
munlock()	1334, 1384
munlockall()	1336, 1385
munmap()	1386, 3601, 3603, 3605, 3608
mutex	70, 3639
attributes	1681
extended attributes	3642
initialization	3655
initialization attributes	1680
performance	1681
MUXID_ALL	369, 1154-1155
MUXID_R	477
mv	3017, 3716, 3773
MX	9
MXX	9
M_	288, 477

M_E	288
M_LN	288
M_LOG10E.....	288
M_LOG2E.....	288
M_PI.....	288
M_SQRT1_2.....	289
M_SQRT2.....	289
name.....	71
name information.....	1057
name space	473, 3567
name space pollution	3567-3568
named STREAM.....	71
NAME_MAX.....	112, 231, 274, 484, 902, 2467, 3097, 3710, 3783
NaN.....	71, 247
NAN.....	289
NaN.....	603, 605, 609, 615
NAN.....	912
NaN.....	912
NAN.....	996
NaN.....	996
NaN arguments	3520
mathematical functions	118
nan().....	1388
nanf()	1388
nanl()	1388
nanosleep()	1389, 3618, 3620, 3622, 3770
native implementation	3498
native language	71
NCCS	415
NDEBUG	223, 480, 608
nearbyint()	1391
nearbyintf().....	1391
nearbyintl()	1391
negative response.....	71
negative_sign.....	155
netent	302
network.....	71
network address	71
network byte order.....	72, 110, 3514
network interfaces.....	524
newgrp.....	2344, 2367, 3023, 3774
newline character	72
newlocale()	1392
news	
rationale for omission.....	3758
NEW_TIME	456, 771-772
nextafter().....	1395
nextafterf()	1395
nextafterl()	1395
nexttoward().....	1395
nexttowardf().....	1395
nexttowardl().....	1395

nftw()	1397
NGROUPS_MAX	275, 1043, 2099, 3026, 3479, 3502, 3710, 3777, 3783
nice	3027, 3772
nice value	72, 3499
nice()	1401, 3612
Ninth Edition UNIX	2536, 2676, 3116
NI_DGRAM	303
NI_NAMEREQD	303
NI_NOFQDN	303
NI_NUMERICHOST	303
NI_NUMERICSCOPE	303
NI_NUMERICSERV	303
nl	3031
NLDLY	416
nlink_t	402
NLn	416
NLSPATH	175, 649
NL_	475
NL_ARGMAX	281, 909, 949, 993, 1003
NL_CAT_LOCALE	312, 649
nl_langinfo()	1403, 3771
nl_langinfo_l()	1403
NL_LANGMAX	281
NL_MSGMAX	281
NL_SETD	312
NL_SETMAX	282
NL_TEXTMAX	282
nm	3035, 3774
noclobber option	3066, 3735
NOEXPR	267
NOFLSH	418
nohup	792, 3040, 3772
non-blocking	72
non-canonical mode input processing	202, 3552
non-local jumps	1982
non-printable	2690, 3223, 3284, 3509
non-spacing characters	72
non-volatile storage	974
normative references	3480
NOSTR	267
NQS	3751
rand48()	749, 1406
ntohl()	1120, 1407
ntohs()	1120, 1407
NUL	73
NULL	346, 425, 699, 732, 741, 1343, 1780
null byte	73
null pointer	73
null string	73
null wide-character code	73
number-sign	73
numerical limits	280

NUM_EMPL.....	1118
NZERO	282, 1073, 1401
n_	475
n_cs_precedes	156
n_sep_by_space	156
n_sign_posn	156
OB	9
object file.....	73, 3035
obsolescent	3481
rationale.....	3481
OCRNL	416
octet	73
od	3044, 3772, 3774
OF	9
OFDEL	416
offset maximum.....	74
off_t	402
OFILL	416
OH	9
OLD_TIME.....	456, 771-772
ONLCR	416
ONLRET	416
ONOCR	416
opaque address.....	74
open a file	1414
open a named semaphore.....	1855
open a shared memory object.....	1935
open file	74
open file description	74, 3499
open file descriptors.....	3737
for reading and writing.....	2363
open mode.....	2697
open()	1408, 3493, 3550, 3601, 3603-3604, 3716, 3768
openat()	1408, 1420
opendir()	838, 1421, 3769
openlog().....	694, 1422, 3774
OPEN_MAX.....	271, 329, 767, 823, 838, 983, 1356, 1460, 1463, 2099, 2163, 3479, 3710
.....	3783-3784
open_memstream()	1418
open_wmemstream()	1418
operand.....	74
operator	74
OPOST	416
optarg.....	1061, 1423
opterr.....	1061, 1423
optind.....	1061, 1423
option	75
ADV	7
BE.....	7
CD.....	7
CPT	7
FD	7

FR.....	7
FSC	8
IP6.....	8
MC1	8
ML	8
MLR.....	8
MON	8
MSG.....	8
MX	9
MXX	9
PIO.....	9
PS	9
RPI	9
RPP	10
RS.....	10
SD	10
SHM	10
SIO	10
SPN.....	10
SS	10
TCT	10
TEF.....	11
TPI.....	11
TPP.....	11
TPS.....	11
TRC.....	11
TRI	11
TRL	11
TSA	12
TSH.....	12
TSP.....	12
TSS.....	12
TYM.....	12
UP	12
UU	12
XSR	13
option definitions	3751
option-argument	75
optional behavior	3785
options	3663
shell and utilities	27
system interfaces	27
optopt.....	1061, 1064, 1423
optstring	1064
OR lists.....	2371, 3742
ordinary identifiers	2529
ORD_CHAR.....	192
orientation	75
orphaned process group	75, 556, 3499, 3582
output device	198, 3548
output file descriptor	
duplication	3737

output mode	3553
output processing.....	3552
Output_Path.....	2438
overrun conditions.....	3688
overrun in dumping trace streams	3688
overrun in trace streams.....	3688
O_	477
O_ constants	
defined in <fcntl.h>	238-239
used in open()	1408
used in posix_openpt()	1450
O_ACCMODE	239, 820
O_APPEND.....	239, 503, 592, 730, 836, 1408, 2310
O_CLOEXEC.....	238, 1408, 3604
O_CREAT.....	238, 714, 1356-1357, 1408, 1846, 1853, 1933-1934, 1936
O_DIRECTORY	238, 1409
O_DSYNC.....	239, 583, 1409, 1772, 2311
O_EXCL.....	239, 1357, 1409, 1853, 1933-1934
O_EXEC	239, 1408
O_NDELAY	2316
O_NOCTTY.....	239, 1409, 1450
O_NOFOLLOW.....	239, 1409
O_NONBLOCK.....	239, 483, 820, 1357, 1409
O_RDONLY	239, 737, 1356, 1408, 1933, 1936
O_RDWR.....	239, 810, 1269, 1356, 1408, 1450, 1933, 1936
O_RSYNC.....	239, 1409, 1772
O_SEARCH	239, 1243, 1316, 1322, 1327, 1408, 1411, 1783, 1817, 2095, 2197
O_SYNC.....	239, 583, 1410, 1772, 2311
O_TRUNC.....	239, 714, 1410, 1934, 1936
O_TTY_INIT	199, 239, 1410
O_WRONLY.....	239, 714, 810, 1269, 1356, 1408
pack	
rationale for omission.....	3758
page.....	75, 3499, 3601, 3604
page size	75
PAGESIZE	271, 505, 1334, 1568, 2099, 3601, 3646, 3783
PAGE_SIZE	272, 2099
paginators	
more	3005
parallel I/O	3646
parameter	76, 3553, 3721
expansion	2354, 3729
positional.....	3721
special	3721
parameters and variables.....	2349
PARENB.....	418
parent directory	76, 3500
parent process.....	76
parent process ID	76
PARMRK	416
PARODD	418

passwd	
rationale for omission.....	3758
passwd file.....	3500
paste	3052 , 3772
patch	3056 , 3773-3774
application.....	3059
file format	3058
filename determination	3059
PATH.....	178 , 699, 793, 3538
path pre x	77
pathchk	3063 , 3773
pathconf().....	902, 1424, 3493, 3709, 3767, 3769
pathname.....	76, 3500
component.....	77
expansion	2360, 3735
incomplete	3494
resolution	111, 2331, 3516
variable values.....	273
pathname manipulation	
basename	2518
dirname.....	2667
pathchk	3063
PATH_MAX	274, 282, 484, 902, 2334, 3103, 3201, 3710, 3783
pattern.....	77
filename expansion	3749
for filename expansion	2383
scanning and processing language.....	2482
pattern matching	2797, 3078, 3354, 3370
de nition	2382
in case statements.....	2372
in shell variables.....	2355
multiple character	3748
multiple characters.....	2383
notation.....	2382, 2803, 3096, 3748
single character.....	2382, 3748
pause().....	1425 , 3581, 3585, 3768
pax.....	3068 , 3773-3774
archive character set encoding/decoding	3102
cpio file data.....	3092
cpio filename.....	3092
cpio header	3090
cpio interchange format	3089
cpio special entries	3092
extended header	3082
extended header file times.....	3085
extended header keyword precedence	3085
list mode format specifications	3076
ustar format.....	3086
ustar interchange format.....	3086
pcat	
rationale for omission.....	3758
pclose()	1426 , 3774

pd_.....	475
PENDIN.....	477
pending error.....	3662
per-thread errno.....	3574
performance enhancements.....	3764
period.....	77
permissions.....	78
perror().....	1428
persistence.....	78
persistent connection (I_PLINK).....	1155
PF_.....	476
pg	
rationale for omission.....	3758
physical write.....	974
ph_.....	475
PID_MAX.....	3690
pid_t.....	402
PIO.....	9
pipe.....	78, 899, 1414, 2314, 3494-3495, 3500
pipe().....	1430 , 3582, 3691, 3768
pipelines.....	2368, 3741
PIPE_BUF.....	274, 902, 2311, 2314, 3710, 3783
PIPE_MAX.....	2316
plain characters.....	2039
PM_STR.....	267
pointer to a function.....	492
pole error.....	117
POLL.....	475
poll().....	1433
POLLERR.....	313, 1433
pollfd.....	313
POLLHUP.....	313, 1433
POLLIN.....	313, 1433
polling.....	78
POLLNVAL.....	313, 1434
POLLOUT.....	313, 1433
POLLPRI.....	313, 1433
POLLRDBAND.....	313, 1433
POLLRDNORM.....	313, 1433
POLLWRBAND.....	313, 1433
POLLWRNORM.....	313, 1433
POLL_.....	475
POLL_ERR.....	337
POLL_HUP.....	337
POLL_IN.....	337
POLL_MSG.....	337
POLL_OUT.....	337
POLL_PRI.....	337
popen().....	1437 , 3771, 3774-3775
portability.....	3482
portability codes.....	3482
portable character set.....	78, 125, 2976, 3521

portable filename.....	79
portable filename character set	79, 3500
positional parameter.....	79, 2349, 3721
positive_sign	155
POSIX conformance	15
POSIX locale.....	136, 3525
POSIX shell and utilities.....	19
POSIX system interfaces	
conformance.....	17
POSIX.1 symbols	472, 3566
POSIX.13.....	3605
POSIX2_BC_BASE_MAX.....	2333-2334, 3777
POSIX2_BC_DIM_MAX	2333-2334, 3777
POSIX2_BC_SCALE_MAX.....	2333-2334, 3777
POSIX2_BC_STRING_MAX.....	2333-2334, 3777
POSIX2_CHAR_TERM.....	19, 27, 3776
POSIX2_COLL_WEIGHTS_MAX	2333-2334, 3777
POSIX2_C_BIND	3711, 3775
POSIX2_C_DEV.....	19, 27, 3711, 3775-3776
POSIX2_EXPR_NEST_MAX	2333-2334, 3777
POSIX2_FORT_DEV	19, 27, 3711, 3776
POSIX2_FORT_RUN.....	19, 27, 3711, 3776
POSIX2_LINE_MAX.....	2333, 2335, 3777
POSIX2_LOCALEDEF	19, 27, 3711, 3773, 3776
POSIX2_PBS.....	19, 28, 3776
POSIX2_PBS_ACCOUNTING	19, 28, 3776
POSIX2_PBS_CHECKPOINT	28, 3776
POSIX2_PBS_LOCATE.....	19, 28, 3776
POSIX2_PBS_MESSAGE.....	19, 28, 3776
POSIX2_PBS_TRACK.....	19, 28, 3776
POSIX2_RE_DUP_MAX.....	3777
POSIX2_SW_DEV.....	19, 28, 3711, 3775
POSIX2_SYMLINKS	902, 2335, 3711
POSIX2_UPE.....	19, 28, 3711, 3775-3776
POSIX2_VERSION	3777
POSIX_.....	474
posix_.....	474
POSIX_ALLOC_SIZE_MIN	274, 902, 3588
POSIX_ASYNCHRONOUS_IO.....	3789
POSIX_BARRIERS	3789
POSIX_CLOCK_SELECTION	3790
POSIX_C_LANG_JUMP	3789
POSIX_C_LANG_MATH.....	3789
POSIX_C_LANG_SUPPORT	3790
POSIX_C_LANG_SUPPORT_R	3790
POSIX_C_LANG_WIDE_CHAR	3790
POSIX_C_LANG_WIDE_CHAR_EXT.....	3790
POSIX_C_LIB_EXT	3790
POSIX_DEVICE_IO	3790
POSIX_DEVICE_IO_EXT.....	3790
POSIX_DEVICE_SPECIFIC	3790
POSIX_DEVICE_SPECIFIC_R.....	3791

POSIX_DYNAMIC_LINKING	3791
posix_fadvise()	1440 , 3587
POSIX_FADV_DONTNEED	240, 1440, 3587
POSIX_FADV_NOREUSE	240, 1440, 3587
POSIX_FADV_NORMAL	240, 1440
POSIX_FADV_RANDOM	240, 1440, 3587
POSIX_FADV_SEQUENTIAL	240, 1440, 3587
POSIX_FADV_WILLNEED	240, 1440, 3587
posix_fallocate()	1442
POSIX_FD_MGMT	3791
POSIX_FIFO	3791
POSIX_FIFO_FD	3791
POSIX_FILE_ATTRIBUTES	3791
POSIX_FILE_ATTRIBUTES_FD	3791
POSIX_FILE_LOCKING	3791
POSIX_FILE_SYSTEM	3791
POSIX_FILE_SYSTEM_EXT	3791
POSIX_FILE_SYSTEM_FD	3791
POSIX_FILE_SYSTEM_GLOB	3791
POSIX_FILE_SYSTEM_R	3791
POSIX_I18N	3791
POSIX_JOB_CONTROL	3791
posix_madvise()	1444 , 3587
POSIX_MADV_DONTNEED	373, 1444, 3587
POSIX_MADV_NORMAL	373, 1444
POSIX_MADV_RANDOM	373, 1444, 3587
POSIX_MADV_SEQUENTIAL	373, 1444, 3587
POSIX_MADV_WILLNEED	374, 1444, 3587
POSIX_MAPPED_FILES	3791
posix_memalign()	1448
POSIX_MEMORY_PROTECTION	3791
posix_mem_offset()	1446 , 3605-3606
POSIX_MULTI_CONCURRENT_LOCALES	3791
POSIX_MULTI_PROCESS	3792
POSIX_MULTI_PROCESS_FD	3792
POSIX_NETWORKING	3792
posix_openpt()	1450
POSIX_PIPE	3792
POSIX_REALTIME_SIGNALS	3792
POSIX_REC_INCR_XFER_SIZE	274, 902, 3588
POSIX_REC_MAX_XFER_SIZE	274, 902, 3588
POSIX_REC_MIN_XFER_SIZE	274, 902, 3588
POSIX_REC_XFER_ALIGN	274, 902, 3588
POSIX_REGEX	3792
POSIX_RE_DUP_MAX	2333, 2335
POSIX_ROBUST_MUTEXES	3792
POSIX_RW_LOCKS	3792
POSIX_SEMAPHORES	3792
POSIX_SHELL_FUNC	3792
POSIX_SIGNALS	3792
POSIX_SIGNALS_EXT	3792
POSIX_SIGNAL_JUMP	3792

POSIX_SINGLE_PROCESS	3793
posix_spawn()	1452, 3694, 3768
posix_spawnattr_destroy()	1468
posix_spawnattr_get_ags()	1470
posix_spawnattr_getpgroup()	1472
posix_spawnattr_getschedparam()	1474
posix_spawnattr_getschedpolicy()	1476
posix_spawnattr_getsigdefault()	1478
posix_spawnattr_getsigmask()	1480
posix_spawnattr_init()	1468, 1482
posix_spawnattr_set_ags()	1470, 1483
posix_spawnattr_setpgroup()	1472, 1484
posix_spawnattr_setschedparam()	1474, 1485
posix_spawnattr_setschedpolicy()	1476, 1486
posix_spawnattr_setsigdefault()	1478, 1487
posix_spawnattr_setsigmask()	1480, 1488
posix_spawnp()	1452, 1489, 3694, 3768
posix_spawn_ le_actions_addclose()	1460
posix_spawn_ le_actions_adddup2()	1463
posix_spawn_ le_actions_addopen()	1460, 1465
posix_spawn_ le_actions_destoy()	1466
posix_spawn_ le_actions_init()	1466
POSIX_SPAWN_RESETIDS	1453, 1470
POSIX_SPAWN_SETPGROUP	1453, 1470, 1472
POSIX_SPAWN_SETSCHEDPARAM	1470, 1474
POSIX_SPAWN_SETSCHEDULER	1453, 1470, 1474, 1476
POSIX_SPAWN_SETSIGDEF	1454, 1470, 1478
POSIX_SPAWN_SETSIGMASK	1470, 1480
POSIX_SPIN_LOCKS	3793
POSIX_SYMBOLIC_LINKS	3793
POSIX_SYMBOLIC_LINKS_FD	3793
POSIX_SYSTEM_DATABASE	3793
POSIX_SYSTEM_DATABASE_R	3793
POSIX_THREADS_BASE	3793
POSIX_THREADS_EXT	3793
POSIX_TIMERS	3793
POSIX_TRACE_ADD_EVENTSET	1529
POSIX_TRACE_ALL_EVENTS	1522
POSIX_TRACE_APPEND	1495, 1515
posix_trace_attr_destroy()	1490
posix_trace_attr_getclockres()	1492
posix_trace_attr_getcreatetime()	1492
posix_trace_attr_getgenversion()	1492
posix_trace_attr_getinherited()	1494
posix_trace_attr_getlogfullpolicy()	1494
posix_trace_attr_getlogsize()	1497
posix_trace_attr_getmaxdatasize()	1497
posix_trace_attr_getmaxsystemeventsz()	1497
posix_trace_attr_getmaxusereventsz()	1497
posix_trace_attr_getname()	1492, 1500
posix_trace_attr_getstreamfullpolicy()	1494, 1501
posix_trace_attr_getstreamsize()	1497, 1502

posix_trace_attr_init()	1490, 1503
posix_trace_attr_setinherited()	1494, 1504
posix_trace_attr_setlogfullpolicy()	1494, 1504
posix_trace_attr_setlogsize()	1497, 1505
posix_trace_attr_setmaxdatasize()	1497, 1505
posix_trace_attr_setname()	1492, 1506
posix_trace_attr_setstreamfullpolicy()	1494, 1507
posix_trace_attr_setstreamsize()	1497, 1508
posix_trace_clear()	1509
posix_trace_close()	1511
POSIX_TRACE_CLOSE_FOR_CHILD	1494
posix_trace_create()	1513
posix_trace_create_withlog()	1513
POSIX_TRACE_ERROR trace event	543
posix_trace_event()	1517
posix_trace_eventid_equal()	1519
posix_trace_eventid_get_name()	1519
posix_trace_eventid_open()	1517, 1521, 3683
posix_trace_eventset_add()	1522
posix_trace_eventset_del()	1522
posix_trace_eventset_empty()	1522
posix_trace_eventset_II()	1522
posix_trace_eventset_ismember()	1522
posix_trace_eventtypelist_getnext_id()	1524
posix_trace_eventtypelist_rewind()	1524
posix_trace_event_info	540
POSIX_TRACE_FILTER trace event	543, 1529
POSIX_TRACE_FLUSH	1495
posix_trace_flush()	1513, 1526
POSIX_TRACE_FLUSHING	539
POSIX_TRACE_FULL	538-540
posix_trace_getnext_event()	1532
posix_trace_get_attr()	1527
posix_trace_get_iterator()	1529
posix_trace_get_status()	1527, 1531
POSIX_TRACE_INHERITED	1494
POSIX_TRACE_LOOP	539, 1494-1495, 1515, 3688
POSIX_TRACE_NOT_FLUSHING	540
POSIX_TRACE_NOT_FULL	538-540
POSIX_TRACE_NOT_FULL	1509
POSIX_TRACE_NOT_TRUNCATED	541, 1533
POSIX_TRACE_NO_OVERRUN	539-540, 1527
posix_trace_open()	1511, 1535
POSIX_TRACE_OVERFLOW trace event	543
POSIX_TRACE_OVERRUN	539-540
POSIX_TRACE_RESUME trace event	543
posix_trace_rewind()	1511, 1535
POSIX_TRACE_RUNNING	538-539, 1538
POSIX_TRACE_SET_EVENTSET	1529
posix_trace_set_filter()	1529, 1536
posix_trace_shutdown()	1513, 1537
POSIX_TRACE_START trace event	543, 1538

posix_trace_start()	1538
posix_trace_status_info	538
POSIX_TRACE_STOP trace event	543, 1538
posix_trace_stop()	1538
POSIX_TRACE_SUB_EVENTSET	1529
POSIX_TRACE_SUSPENDED	538-539, 1538
POSIX_TRACE_SYSTEM_EVENTS	1522
posix_trace_timedgetnext_event()	1532, 1540
posix_trace_trid_eventid_open()	1519, 1541
POSIX_TRACE_TRUNCATED_READ	541, 1533
POSIX_TRACE_TRUNCATED_RECORD	541, 1533
posix_trace_trygetnext_event()	1532, 1542
POSIX_TRACE_UNTIL_FULL	539, 1494-1495, 1514
POSIX_TRACE_USER_EVENT_MAX	1517
POSIX_TRACE_WOPID_EVENTS	1522
POSIX_TYPED_MEM_ALLOCATE	374, 1338-1339, 1446, 1543, 1545
POSIX_TYPED_MEM_ALLOCATE_CONTIG	374, 1338-1339, 1446, 1543, 1545
posix_typed_mem_get_info()	1543, 3605
posix_typed_mem_info	374
POSIX_TYPED_MEM_MAP_ALLOCATABLE	374, 1386, 1545
posix_typed_mem_open()	1545, 3605
POSIX_USER_GROUPS	3793
POSIX_USER_GROUPS_R	3793
POSIX_VERSION	3783
POSIX_WIDE_CHAR_DEVICE_IO	3793
post-mortem filtering of trace event types	3685
pow()	1548
powf()	1548
powl()	1548
pr	3107, 3772, 3774
pread()	1551, 1771, 3647
preallocation	79
predefined stream	
standard error	498
standard input	498
standard output	498
preempted process (or thread)	79
preempted thread	1620
previous job	79
PRI	477
print-related commands	
fold	2806
lp	2918
pr	3107
printable character	80
printable file	80
printf	3112, 3771-3772
printf()	909, 1552
printing	3766
priority	80, 500
Priority	2439

priority	
band.....	80
inversion.....	80
scheduling.....	80
priority-based scheduling.....	80
PRIO_.....	475
PRIO_ constants	
defined in <sys/resource.h>.....	378
PRIO_INHERIT.....	1676
PRIO_PGRP.....	378, 1073
PRIO_PROCESS.....	378, 1073
PRIO_USER.....	378, 1073
privilege.....	81, 2997, 3029, 3511
process.....	81
attributes.....	2327
concurrent execution.....	899
ID.....	82, 2327
ID reuse.....	113, 3518
ID, 1.....	556
ID, rationale.....	3690
lifetime.....	82, 3501
memory locking.....	82
scheduling.....	506, 3610, 3768
setting real and effective user IDs.....	1918
single-threaded.....	899
termination.....	82, 3501
virtual time.....	83
process creation.....	899
process group.....	81, 3549
concepts in job control.....	3494
ID.....	81, 2327, 3494, 3549-3550
leader.....	81
lifetime.....	81, 3550
orphaned.....	556, 3499, 3582
termios.....	200
process group ID.....	1069, 1910, 1922
process lifetime.....	1228
process management.....	3764, 3768
process shared memory.....	1681
process status report.....	3123
process synchronization.....	1681
process termination.....	555
process-to-process communication.....	82
prof	
rationale for omission.....	3758
pro ling.....	3775
program.....	83
programming manipulation.....	3678
prompting.....	3727
protocol.....	83, 3662
protoent.....	302
PROT_.....	475

PROT_EXEC.....	373, 1339, 1348
PROT_NONE	373, 505, 1338-1339, 1348
PROT_READ.....	373, 1339, 1348
PROT_READ constants	
in <sys/mman.h>	373
PROT_WRITE.....	373, 1339-1340, 1343, 1348
prs	3118
PS	9
ps	3123, 3772, 3774
pselect()	1553
pseudo-random sequence generation functions	1768
pseudo-terminal	83
psiginfo().....	1558
psignal()	1558
PST8PDT.....	2185
ps_	475
pthread.....	3686
PTHREAD_	475
pthread_	475
pthread_atfork().....	1560
pthread_attr_destroy().....	1563
pthread_attr_getdetachstate().....	1566
pthread_attr_getguardsize()	1568, 3646
pthread_attr_getinheritsched().....	1571
pthread_attr_getschedparam().....	1573
pthread_attr_getschedpolicy().....	1575
pthread_attr_getscope().....	1577
pthread_attr_getstack().....	1579
pthread_attr_getstackaddr()	3693
pthread_attr_getstacksize().....	1582
pthread_attr_init()	1563, 1584
pthread_attr_setdetachstate()	1566, 1585
pthread_attr_setguardsize().....	1568, 1586, 3646
pthread_attr_setinheritsched()	1571, 1587
pthread_attr_setschedparam().....	1573, 1588
pthread_attr_setschedpolicy()	1575, 1589
pthread_attr_setscope()	1577, 1590
pthread_attr_setstack()	1579, 1591
pthread_attr_setstackaddr().....	3693
pthread_attr_setstacksize()	1582, 1592
pthread_barrierattr_destroy().....	1597
pthread_barrierattr_getpshared()	1599
pthread_barrierattr_init()	1597, 1601
pthread_barrierattr_setpshared().....	1599, 1602
pthread_barrier_destroy().....	1593
pthread_barrier_init()	1593
PTHREAD_BARRIER_SERIAL_THREAD	315, 1595, 3637
pthread_barrier_wait().....	1595, 3637, 3658
pthread_cancel()	1603
PTHREAD_CANCELED.....	315, 521, 1640
PTHREAD_CANCEL_ASYNCHRONOUS	315, 517, 1727
PTHREAD_CANCEL_DEFERRED	315, 517, 521, 786, 1617, 1727

PTHREAD_CANCEL_DISABLE	315, 517, 521, 1727
PTHREAD_CANCEL_ENABLE	315, 517, 521, 1727
PTHREAD_CANCEL_ENABLED	786
pthread_cleanup_pop()	1605
pthread_cleanup_push()	1605
pthread_condattr_destroy()	1624
pthread_condattr_getclock()	1626
pthread_condattr_getpshared()	1628
pthread_condattr_init()	1624, 1630
pthread_condattr_setclock()	1626, 1631
pthread_condattr_setpshared()	1628, 1632
pthread_cond_broadcast()	1610
pthread_cond_destroy()	1613
pthread_cond_init()	1613, 3634
PTHREAD_COND_INITIALIZER	315, 1613
pthread_cond_signal()	1610, 1616
pthread_cond_timedwait()	1617 , 3575, 3621, 3643, 3778
pthread_cond_wait()	1617, 3575, 3592, 3643
pthread_create()	1633 , 3633-3634
PTHREAD_CREATE_DETACHED	315, 490, 1566, 3656
PTHREAD_CREATE_JOINABLE	315, 490, 786, 1566, 1650
PTHREAD_DESTRUCTOR_ITERATIONS	272, 1647, 1652, 2099, 3784
pthread_detach()	1636 , 3656
pthread_equal()	1638
pthread_exit()	1639
PTHREAD_EXPLICIT_SCHED	315, 1571
pthread_getconcurrency()	1641 , 3646
pthread_getcpuclockid()	1643 , 3623, 3625
pthread_getschedparam()	1644
pthread_getspeci c()	1647
PTHREAD_INHERIT_SCHED	315, 1571
pthread_join()	1649 , 3575, 3656
PTHREAD_KEYS_MAX	272, 1652, 2099, 3784
pthread_key_create()	1652 , 3636
pthread_key_delete()	1655
pthread_kill()	1657
pthread_mutexattr_destroy()	1680
pthread_mutexattr_getprioceiling()	1685
pthread_mutexattr_getprotocol()	1687
pthread_mutexattr_getpshared()	1690
pthread_mutexattr_getrobust()	1692
pthread_mutexattr_gettype()	1694 , 3644
pthread_mutexattr_init()	1680, 1696
pthread_mutexattr_setprioceiling()	1685, 1697
pthread_mutexattr_setprotocol()	1687, 1698
pthread_mutexattr_setpshared()	1690, 1699
pthread_mutexattr_setrobust()	1692, 1700
pthread_mutexattr_settype()	1694, 1701, 3644
pthread_mutex_consistent()	1659
PTHREAD_MUTEX_DEFAULT	315, 1671, 1694, 3643
pthread_mutex_destroy()	1661
PTHREAD_MUTEX_ERRORCHECK	315, 1667, 1671, 1694, 3643

pthread_mutex_getprioceiling()	1667
pthread_mutex_init()	1661, 1670, 3634
PTHREAD_MUTEX_INITIALIZER	315, 1661
pthread_mutex_lock()	1671, 3575, 3643, 3658
PTHREAD_MUTEX_NORMAL	315, 1671, 1694, 3643
PTHREAD_MUTEX_RECURSIVE	111, 315, 1671, 1694, 3643
PTHREAD_MUTEX_ROBUST	1692
pthread_mutex_setprioceiling()	1667, 1675
PTHREAD_MUTEX_STALLED	1692
pthread_mutex_timedlock()	1676, 3622
pthread_mutex_trylock()	1671, 1679, 3643
pthread_mutex_unlock()	1671, 1679, 3643
pthread_once()	1702
PTHREAD_ONCE_INIT	315, 1702
PTHREAD_PRIO_INHERIT	315, 1687
PTHREAD_PRIO_NONE	315, 1667, 1687
PTHREAD_PRIO_PROTECT	315, 1672, 1687
PTHREAD_PROCESS_PRIVATE	315, 1681, 3645
PTHREAD_PROCESS_SHARED	315, 1599, 1628, 1681, 1690, 1722, 1738, 3645
pthread_rwlockattr_destroy()	1720, 3645
pthread_rwlockattr_getpshared()	1722, 3645
pthread_rwlockattr_init()	1720, 1724, 3644
pthread_rwlockattr_setpshared()	1722, 1725, 3645
pthread_rwlock_destroy()	1704
pthread_rwlock_init()	1704, 3645
PTHREAD_RWLOCK_INITIALIZER	315, 3645
pthread_rwlock_rdlock()	1707, 3645
pthread_rwlock_t	3644
pthread_rwlock_timedrdlock()	1710
pthread_rwlock_timedwrlock()	1712
pthread_rwlock_tryrdlock()	1707, 1714, 3645
pthread_rwlock_trywrlock()	1715, 3645
pthread_rwlock_unlock()	1717, 3645, 3660
pthread_rwlock_wrlock()	1715, 1719, 3645
PTHREAD_SCOPE_PROCESS	315, 515, 1577
PTHREAD_SCOPE_SYSTEM	315, 515, 1577
pthread_self()	1726, 3635
pthread_setcancelstate()	1727
pthread_setcanceltype()	1727
pthread_setconcurrency()	1641, 1729, 3645-3646
pthread_setprio()	3654
pthread_setschedparam()	1644, 1730, 3654
pthread_setschedprio()	1731
pthread_setspeci c()	1647, 1733, 3636
pthread_sigmask()	1734
pthread_spin_destroy()	1738
pthread_spin_init()	1738
pthread_spin_lock()	1740, 3638, 3658
pthread_spin_trylock()	1740, 3638
pthread_spin_unlock()	1742
PTHREAD_STACK_MIN	272, 1579, 1582, 2099, 3784
pthread_testcancel()	1727, 1744

PTHREAD_THREADS_MAX	272, 1633, 2099, 3784
PTRDIFF_MAX.....	352
PTRDIFF_MIN.....	352
ptsname()	1745
public locale	2903
putc()	1747 , 3648, 3769
putchar()	1749 , 3769
putchar_unlocked()	1013, 1750
putc_unlocked().....	1013, 1748
putenv().....	1751
putmsg().....	1753
putpmsg()	1753
puts()	1757
pututxline().....	771, 1759
putwc().....	1760
putwchar()	1761
PWD	178
pwd	2344, 2367, 3130, 3773
pwrite().....	1762 , 2310, 3647
pw_.....	475
p_.....	475
P_.....	476
P_ALL	409, 2236
p_cs_precedes	156
P_PGID	409, 2236
P_PID	409, 2236
p_sep_by_space.....	156
p_sign_posn	156
P_tmpdir.....	356
qalter	3133
qdel.....	3143
qhold	3146
qmove	3149
qmsg.....	3152
qrerun.....	3155
qrls.....	3158
qselect.....	3161
qsig	3170
qsort()	1763
qstat	3173
qsub	3178
queue a signal to a process	1980
queue batch job request.....	2445
queuing of waiting threads.....	3660
quiet NaN.....	247
quote removal.....	2360, 3735
QUOTED_CHAR	192
quoting.....	2346, 3718
radix character	83
RADIXCHAR.....	267
raise()	1765
rand()	1767 , 3658

random()	1141, 1770
RAND_MAX	359, 1767
rand_r()	1767
range error	118
result overflows	118
result underflows	118
RCS	
rationale for omission	3758
RE	
bracket expression	3542
grammar	3547
read	2344, 2367, 3191, 3717, 3771
read lock	3644
read()	1771, 3494, 3551-3552, 3572, 3581-3582, 3585, 3595-3596, 3600-3601, 3646 3657, 3690, 3769
read-only file system	83
read-write attribute	3644
read-write lock	83, 3644
readdir()	1778, 3769
readdir_r()	1778
reading an active trace stream	3688
reading data	3551
readlink()	1783, 3768
readlinkat()	1783
readonly	2404
readv()	1786
real group ID	84, 2327
real time	84
real user ID	84, 572, 1227, 2327
realloc()	1788
realpath()	1790, 3768
realtime	22
REALTIME	297, 829, 1334, 1336, 1350-1351, 1353, 1356, 1359, 1362, 1364, 1368, 1833-1837 1839, 1933, 1938
realtime	3587
realtime signal delivery	3578
realtime signal extension	84
realtime signal generation	3578
realtime signals	3593
REALTIME THREADS	24
realtime threads	24
REALTIME THREADS	1571, 1575, 1577, 1644, 1667, 1685, 1687, 1731
record	84
recv()	1793
recvfrom()	1796
recvmsg()	1799
red	
rationale for omission	3758
redirect input	2361, 3736
redirect output	2361, 3736
redirection	84, 2360, 3735
redirection operator	85

referenced shared memory object.....	85
references.....	3480
refresh.....	85
regcomp().....	1802 , 3774
regerror().....	1802, 3774
regexec().....	1802, 3774
regfree().....	1802, 3774
region.....	85
register fork handlers.....	1560
REGTYPE.....	413
regular expressions.....	85, 2491, 2609, 2679, 2734, 2775, 2800, 2842, 2888, 3009, 3019
.....	3031, 3075, 3199, 3218, 3392, 3449, 3539
basic.....	183
definitions.....	3540
extended.....	188
general requirements.....	3541
grammar.....	192, 3546
related to shell patterns.....	2382
regular file.....	85, 3501
REG.....	475
REG_ constants	
defined in <regex.h>.....	323
error return values of regcomp.....	1804
used in regcomp.....	1802
REG_BADBR.....	324, 1804
REG_BADPAT.....	323, 1804
REG_BADRPT.....	324, 1804
REG_EBRACE.....	324, 1804
REG_EBRACK.....	324, 1804
REG_ECOLLATE.....	323, 1804
REG_ECTYPE.....	323, 1804
REG_EESCAPE.....	323, 1804
REG_EPAREN.....	324, 1804
REG_ERANGE.....	324, 1804
REG_ESPACE.....	324, 1804
REG_ESUBREG.....	323, 1804
REG_EXTENDED.....	323, 1802
REG_ICASE.....	323, 1802
REG_NEWLINE.....	323, 1802
REG_NOMATCH.....	323, 1804
REG_NOSUB.....	323, 1802
REG_NOTBOL.....	323, 1803
REG_NOTEOL.....	323, 1803
rejected utilities.....	3756
relational database operator.....	2874
relative pathname.....	86, 111
release batch job request.....	2446
relocatable file.....	86
relocation.....	86
remainder().....	1809
remainderf().....	1809
remainderl().....	1809

remove a directory	1826, 3205
remove a directory entry	2200
remove a file	3198
remove()	1811
remque()	1143, 1813
remquo()	1814
remquof()	1814
remquol()	1814
rename a file	1819
rename()	1816 , 3769
renameat()	1816
renice	3194 , 3772
replenishment period	3614
requested batch service	86, 2442
requirements	15
rerun batch job request	2447
Rerunable	2439
reserved words	2349, 3721
Resource_List	2439
result overflows	118
result underflows	118
return	2407
rewind()	1821
rewinddir()	1822 , 3769
re_	475
RE_DUP_MAX	275, 2099, 2334, 3711
rindex()	3693
rint()	1823
rintf()	1823
rintl()	1823
rlimit	378
RLIMIT_	475
RLIMIT_AS	379, 1087
RLIMIT_CORE	378, 1086
RLIMIT_CPU	378, 1086
RLIMIT_DATA	379, 1086
RLIMIT_FSIZE	379, 1086
RLIMIT_NOFILE	379, 1086, 1088
RLIMIT_STACK	379, 1087
rlim_	475
RLIM_	477
RLIM_INFINITY	378, 1086-1087
RLIM_SAVED_CUR	378, 1087
RLIM_SAVED_MAX	378, 1087
rm	3198 , 3716, 3773
rm del	3203
rm dir	3205 , 3773
rm dir()	1825 , 3574, 3769
RMSGD	368, 1149
RMSGN	369, 1149
RNORM	369, 1149
robust mutex	86, 514, 1665, 3640

root directory	86, 2327, 3501, 3517
root file system.....	3501
root of a file system	3501
round robin	508
round()	1828
roundf().....	1828
roundl().....	1828
routing	524, 3662
RPI.....	9
RPP.....	10
RPROTDAT	369, 1149
RPROTDIS.....	369, 1149
RPROTNORM.....	369, 1149
RS.....	10
rsh	
rationale for omission.....	3758
RS_HIPRI.....	368, 1053, 1148, 1753
RTLD_.....	475
RTLD_DEFAULT.....	746
RTLD_GLOBAL	233, 744
RTLD_LAZY	233, 743
RTLD_LOCAL	233, 744
RTLD_NEXT	746
RTLD_NOW.....	233, 744
RTSIG_MAX.....	272, 333, 2099, 3784
runnable process (or thread).....	86
running process (or thread).....	87
runtime values	
increasable.....	274
invariant	270
rusage.....	378
RUSAGE_.....	475
RUSAGE_CHILDREN.....	378, 1089
RUSAGE_SELF.....	378, 1089
ru.....	475
R_ANCHOR	192
R_OK.....	441
s6_.....	475
sact.....	3208
same le().....	3689
saved resource limits	87
saved set-group-ID.....	87, 2327
saved set-user-ID.....	87, 2327
SA_.....	475
sa_.....	475-476
SA_macros	
declared in <signal.h>	335
SA_NOCLDSTOP	335, 491, 1950, 1955, 3495
SA_NOCLDWAIT	335, 1089, 1952
SA_NODEFER.....	335, 1952
SA_ONSTACK.....	335, 785, 1951
SA_RESETHAND	335, 1951-1952, 3692

SA_RESTART	335, 1556, 1951, 1967, 3692
SA_SIGINFO	335, 1950-1951, 1954, 1979, 3580
scalb()	3693
scalbln()	1829
scalblnf()	1829
scalblnl()	1829
scalbn()	1829
scalbnf()	1829
scalbnl()	1829
scandir()	597, 1831
scanf()	949, 1832
sccs	3211
SCCS commands	
admin	2454
delta	2650
get	2823
prs	3118
rmdel	3203
sact	3208
scs	3211
unget	3343
val	3372
what	3437
SCHAR_MAX	280
SCHAR_MIN	280-281
schedule alarm	595
scheduling	87
scheduling allocation domain	87, 3652
scheduling contention scope	87, 3652-3653
scheduling documentation	517, 3653
scheduling policy	88, 113, 3518
round robin	508
SCHED_	475
sched_	475
SCHED_FIFO	325, 504, 507, 516, 785, 897, 1073, 1401, 1573, 1575, 1644, 1685, 1707, 1856
.....	3770
sched_getparam()	1834
sched_getscheduler()	1835
sched_get_priority_max()	1833
sched_get_priority_min()	1833
SCHED_OTHER	325, 507, 510, 1073, 1575, 1644
SCHED_RR	325, 504, 507-508, 516, 785, 897, 1073, 1401, 1573, 1575, 1644, 1707, 1856
.....	3770
sched_rr_get_interval()	1836
sched_setparam()	1837
sched_setscheduler()	1839
SCHED_SPORADIC	325, 504, 507, 509, 785, 1707, 1856, 3770
sched_yield()	1841
SCM_	476
SCM_RIGHTS	387
SCN	477
scope	3477

screen	88
scroll	88
SD	10
sdb	
rationale for omission	3759
sdiff	
rationale for omission	3759
search pattern.....	2621
seconds since the Epoch	113, 3518
security considerations.....	555, 672, 968, 1227, 1909, 3488, 3492, 3497, 3510-3511, 3551
security, monolithic privileges	3488
sed	3216, 3772-3773
addresses	3218
editing commands.....	3218
regular expressions	3218
seed48()	749, 1842
seekdir()	1843
SEEK_.....	477
SEEK_CUR.....	238, 355, 443, 822, 957, 1292
SEEK_END.....	238, 355, 443, 822, 881, 957, 1292
SEEK_GET.....	1821
SEEK_SET.....	238, 355, 443, 503, 585, 592, 822, 957, 1292
SEGV_.....	475
SEGV_ACCERR.....	337
SEGV_MAPERR.....	337
select batch jobs request	2447
select()	1553, 1845
sem	476
sem*().....	3586
semaphore	88, 114, 3519, 3591, 3770
lock operation	114
unlock operation.....	114
semctl().....	1867, 3587
semget().....	1870, 3587
semid.....	502
semop().....	1872, 3587
SEM_.....	475
sem_.....	475
SEM_.....	476
sem_close()	1846
sem_destroy().....	1848
SEM_FAILED.....	329, 1854-1855
sem_getvalue().....	1849
sem_init()	1851, 3591
SEM_NSEMS_MAX.....	272, 1851, 2100, 3784
sem_open()	1853, 3591
sem_perm.....	502
sem_post().....	1856
sem_timedwait().....	1858, 3622
sem_trywait()	1862, 3575, 3593
SEM_UNDO.....	382, 1872
sem_unlink()	1864

SEM_VALUE_MAX	272, 1851, 1853, 2100, 3784
sem_wait()	1862, 1866, 3575, 3593
send()	1877
sendmsg()	1880
sendto()	1884
sequential lists	2370, 3742
servent.....	302
server shutdown request.....	2448
server status request	2448
service name	934
session.....	89, 556, 1227, 1910, 1922, 3495, 3499, 3550
session leader.....	89
session lifetime	89
session membership.....	2327
set.....	2409 , 3727
set cancelability state	1728
set file creation mask.....	2189
set process group ID for job control	1909
set-group-ID	555, 668, 792, 827, 2327, 2611
set-user-ID	555, 792, 1019, 1227, 2327, 2577, 2611
set-user-ID scripts	3240
SETALL	382, 1867, 1870
setbuf()	1888
setegid().....	1890
setenv().....	1891
seteuid()	1893
setgid().....	1894 , 3502
setgrent().....	758, 1896, 3510
sethostent()	760, 1897
setitimer().....	1047, 1898
setjmp()	1899 , 3771
setkey().....	1901
setlocale()	1902 , 3771
extensions to	1903
setlogmask()	694, 1907, 3774
setnetent()	762, 1908
setpgid()	1909 , 3494-3496, 3549-3550
setpgrp().....	1912
setpriority().....	1073, 1913, 3612
setprotoent()	764, 1914
setpwent().....	766, 1915, 3510
setregid().....	1916
setreuid().....	1918
setrlimit()	1086, 1920, 3716
setservent()	769, 1921
setsid()	1922 , 3549
setsockopt().....	1924
setstate()	1141, 1926
setuid()	1927 , 3502
setutxent().....	771, 1930
SETVAL.....	382, 1867, 1870
setvbuf()	1931

sh.....	3226, 3773, 3780
command history list.....	3230
command line editing.....	3230
vi line editing command mode.....	3232
vi line editing insert mode.....	3231
vi-mode command line editing.....	3231
shall.....	6, 3480
rationale.....	3480
shar	
rationale for omission.....	3759
shared memory.....	3602
shared memory object.....	89
shell.....	89
SHELL.....	178
shell.....	556, 791, 1051, 1069, 1227, 1910, 2232, 3494-3497
SHELL.....	3539
shell.....	3549, 3551, 3576, 3582
commands.....	2365, 3738
errors.....	3737
execution environment.....	2381, 2460, 3193, 3330, 3721, 3747
grammar.....	2375, 3746
grammar rules.....	2375, 3747
grammar, lexical conventions.....	2375, 3746
introduction.....	2345
job control.....	1227, 3494, 3576, 3582
login.....	791, 1051
variables.....	2351, 3726
shell command language.....	2345
alias substitution.....	2348
appending redirected output.....	2361
arithmetic expansion.....	2358
command substitution.....	2357
compound commands.....	2371
consequences of shell errors.....	2363
double-quote.....	2346
duplicating an input file descriptor.....	2363
duplicating an output file descriptor.....	2363
escape character (backslash).....	2346
exit status and errors.....	2363
exit status for commands.....	2364
field splitting.....	2359
function definition command.....	2374
grammar.....	2375
here-document.....	2362
introduction.....	2345
lists.....	2369
open file descriptors for reading and writing.....	2363
parameter expansion.....	2354
parameters and variables.....	2349
pathname expansion.....	2360
pattern matching notation.....	2382
patterns matching a single character.....	2382

patterns matching multiple characters	2383
patterns used for filename expansion	2383
pipelines	2368
positional parameters	2349
quote removal	2360
quoting	2346
redirecting input	2361
redirecting output	2361
redirection	2360
reserved words	2349
shell commands	2365
shell execution environment	2381
shell grammar lexical conventions	2375
shell grammar rules	2375
shell variables	2351
signals and error handling	2381
simple commands	2365
single-quote	2346
special built-in utilities	2384
special parameter	2350
tilde expansion	2354
token recognition	2347
word expansions	2353
shell script	90
exec	791
shell, the	89
Shell_Path_List	2439
shift	2416
shl	
rationale for omission	3759
SHM	10 , 476
shm	476
shm*()	3586
shmat()	1940
shmctl()	1942 , 3587
shmdt()	1944 , 3587
shmget()	1946
shmid	502
SHMLBA	384, 1940
shm_	475
SHM_	476
shm_open()	1933 , 3601, 3603-3604
shm_perm	502
SHM_RDONLY	384, 1940
SHM_RND	384, 1940
shm_unlink()	1938 , 3603-3604
should	6, 3480
rationale	3480
SHRT_MAX	281
SHRT_MIN	281
shutdown()	1948
SHUT_	476

SHUT_RD.....	389
SHUT_RDWR.....	389
SHUT_WR.....	389
SIGABRT.....	333, 565, 3508, 3576
sigaction().....	1950, 3578, 3580
sigaddset().....	1957
SIGALRM.....	333, 595, 1047, 1998
sigaltstack().....	1958
SIGBUS.....	333, 337, 506, 1340, 1343, 1734, 3508, 3576
SIGCANCEL.....	1603
SIGCHLD.....	333, 337, 695, 1089, 1115, 1950, 1955, 1964, 2107, 2237, 3495, 3578, 3581-3582
SIGCLD.....	1955, 3581
SIGCONT.....	333, 495, 554, 556, 1226-1227, 2700, 3495, 3578, 3581-3582
sigdelset().....	1960
sigemptyset().....	1961
SIGEMT.....	3576
SIGEV_.....	475
sigev_.....	475
SIGEV_NONE.....	332, 489, 504, 3578
SIGEV_SIGNAL.....	332, 489, 2151, 3579
SIGEV_THREAD.....	332, 490, 1249, 3579
sig_illset().....	1963
SIGFPE.....	333, 337, 1734, 1971, 3508, 3576, 3578
sighold().....	1964
SIGHUP.....	333, 554, 556, 688, 2678, 2700, 3375, 3416, 3582
sigignore().....	1964
SIGILL.....	333, 337, 1734, 1971, 3508, 3576
siginfo_t.....	336
SIGINT.....	333, 899, 2107, 2381, 2652, 2678, 2699, 3428, 3496, 3650, 3747
siginterrupt().....	1967
SIGIOT.....	3576
sigismember().....	1969
SIGKILL.....	333, 1227, 1950, 1954-1955, 1964, 3576, 3578, 3582
siglongjmp().....	1970, 3572, 3583, 3771
signal.....	90, 488, 3501, 3663, 3747
acceptance.....	3577
actions.....	3581
concepts.....	3575
delivery.....	488, 3577
error handling.....	2381
generation.....	488, 3577
names.....	3575
realtime delivery.....	489
realtime generation.....	489
stack.....	90
signal batch job request.....	2448
signal handler.....	1971
signal processes.....	2879
signal().....	1971, 3575, 3578
signaling NaN.....	247
signbit().....	1974
siggam.....	1241

signgam()	1975
sigpause()	1964, 1976
sigpending()	1977
SIGPIPE	333, 818, 860, 923, 928, 958, 961, 1754, 2313, 3508, 3574
SIGPOLL	333, 337, 688, 1147-1148
sigprocmask()	1734, 1978, 3577
SIGPROF	333, 1047
sigqueue()	1979
SIGQUEUE_MAX	272, 1979, 2100
SIGQUIT	333, 2107, 2381, 2678, 3747
sigrelse()	1964, 1981
SIGRTMAX	333, 489-490, 1953, 1979, 1986, 1990, 3580
SIGRTMIN	333, 489-490, 1953, 1979, 1986, 1990, 3580
SIGSEGV	333, 337, 505, 1087, 1386, 1568, 1734, 1971, 3508, 3576
sigset()	1964, 1981
sigsetjmp()	1982, 3771
sigset_t	3575
SIGSTKSZ	335, 1958
SIGSTOP	333, 489, 1950, 1955, 1964, 3582
sigsuspend()	1984, 3581, 3585
SIGSYS	333, 3576
SIGTERM	333, 2700, 3576
sigtimedwait()	1986, 3575, 3595, 3621
SIGTRAP	333, 337, 3576
SIGTSTP	333, 489, 2745, 3496, 3582
SIGTTIN	333, 489, 863, 869, 1773, 3496, 3551, 3582
SIGTTOU	333, 489, 817, 859, 923, 927, 958, 960, 2118, 2120, 2122, 2129, 2131-2132, 2134 2313, 3495, 3551, 3582
SIGURG	333, 1148
SIGUSR1	333, 3576
SIGUSR2	333, 3576
SIGVTALRM	333, 1047
sigwait()	1990, 3575, 3657
sigwaitinfo()	1986, 1992, 3575, 3595
sigwait_multiple()	3577
SIGXCPU	333, 1086
SIGXFSZ	333, 1086, 2178
SIG_	477
SIG_ATOMIC_MAX	352
SIG_ATOMIC_MIN	352
SIG_BLOCK	335, 1734
SIG_DFL	332, 491, 785, 1087, 1950, 1952, 1971-1972, 3577-3578, 3581
SIG_ERR	332, 1972
SIG_HOLD	332, 1964
SIG_IGN	332, 491, 785, 792, 1089, 1950, 1971, 2381, 3495, 3577-3578, 3581, 3584
SIG_SETMASK	335, 1734
SIG_UNBLOCK	335, 1734
simple commands	2365, 3738
sin()	1993
sin6_	475
sinf()	1993
single-quote	90, 2346, 3718

single-threaded process.....	90
single-threaded program	90
sinh().....	1995
sinhf()	1995
sinhl().....	1995
sinl().....	1993, 1997
sin_	475
SIO	10
SIOCATMARK	2002
sival_	475
size	
rationale for omission.....	3759
SIZE_MAX	352
size_t	402
SI_	475
si_	475
SI_ASYNCIO.....	337, 492
SI_MESGQ.....	337, 492
SI_QUEUE.....	337, 492
SI_TIMER	337, 492
SI_USER.....	337, 492, 3580
slash.....	91
sleep	3244 , 3771
sleep()	1998 , 3583, 3585, 3768, 3770
SLR(1) grammars.....	3468
sl_	475
SND	475
SNDTIMEO	529
SNDZERO	369, 1151
snprintf()	909, 2001
SO	476
sockaddr_in.....	306
sockaddr_in6.....	306
socketmark().....	2002
socket	91, 523, 3661
address.....	91
address families.....	523
addressing	524
asynchronous errors	527
connection indication queue.....	527
I/O mode.....	525, 3662
Internet Protocols	532, 3663
IPv4.....	532, 3663
IPv6.....	532, 3663
local UNIX connection.....	3663
local UNIX connections.....	531
options	528
out-of-band data.....	527
out-of-band data state.....	3663
owner	526, 3662
pending error	526
protocols	524

queue limit	3662
queue limits.....	526
receive queue	526, 3663
signals	527
types.....	524, 3662
socket()	2004
socketpair()	2006 , 3691
SOCK_.....	477
SOCK_DGRAM.....	387, 532, 2004, 2006
SOCK_RAW	387, 532
SOCK_SEQPACKET	387, 532, 2004, 2006
SOCK_STREAM.....	387, 532, 2004, 2006
soft limit.....	91
software development.....	3766, 3774
SOL_SOCKET	387
SOMAXCONN	388
sort	3247 , 3772-3773
source code.....	91
SO_ACCEPTCONN.....	388, 529, 1924
SO_BROADCAST	388, 529
SO_DEBUG	388, 529
SO_DONTROUTE	388, 529
SO_ERROR.....	388, 529, 1924
SO_KEEPAIVE	388, 529
SO_LINGER.....	388, 529
SO_OOBINLINE	388, 529
SO_RCVBUF	388, 529
SO_RCVLOWAT	388, 529
SO_RCVTIMEO.....	388, 529, 1924
SO_REUSEADDR.....	388, 529
SO_SNDBUF	388, 529
SO_SNDLOWAT	388, 529
SO_SNDTIMEO.....	388, 1924
SO_TYPE	388, 529, 1924
space character.....	92
spawn.....	92
spawn example.....	3694
special built-in	92, 2600, 3029, 3042, 3131, 3240, 3299, 3318, 3744
special built-in utilities	2384, 3749
break.....	2386, 2391
characteristics.....	2384
colon.....	2389
dot.....	2393
eval	2395
exec.....	2397
exit	2399
export.....	2401
readonly.....	2404
return.....	2407
set.....	2409
shift.....	2416
times	2418

trap	2420
unset	2424
special characters.....	3552
special control character.....	3554
special parameter	92, 2350, 3721
special targets.....	2974
specific implementation	3493
SPEC_CHAR.....	193
spell	
rationale for omission.....	3759
spin lock	92, 3638-3639
split.....	3255 , 3772
split les	
csplit.....	2617
split.....	3255
SPN.....	10
spoo ng.....	2405
sporadic server	92
sporadic server policy	
execution capacity	509
replenishment period	509
scheduling	3614
sprintf()	909, 2008
spurious wakeup	1611
sqrt()	2009
sqrtf()	2009
sqrtl()	2009
rand()	1767, 2011
rand48()	749, 2012
random().....	1141, 2013
SS	10
sscanf()	949, 2014
SSIZE_MAX	281, 404, 1359, 1375, 1771, 1783, 2041, 2310, 3690, 3784
ssize_t.....	402
SS_	475
ss_	475-476
SS_DISABLE	335, 1958-1959
SS_ONSTACK.....	335, 1958
SS_REPL_MAX.....	272, 3617
stack size.....	1563
stack_t	335
standard error	92, 2360
standard I/O stream.....	495, 3585
standard input	92, 2360
standard output.....	93, 2360
standard utilities.....	93
START	2120
stat	3716
stat data structure.....	392
stat()	965, 2015, 3491, 3601, 3716, 3768
state-dependent character encoding	3523
status information	3691

statvfs()	971, 2016, 3716
stderr	356, 2017, 3585
STDERR_FILENO	447, 2017
stdin	356, 2017, 3585
STDIN_FILENO	447, 1437, 2017
stdio	
locking functions	875
with explicit client locking	1013
stdout	356, 2017, 3585
STDOUT_FILENO	447, 1437, 2017
STOP	2120
stpcpy()	2019, 2029
stpncpy()	2020, 2058
STR	477
strbuf	366
strcasecmp()	2021
strcasecmp_l()	2021
strcat()	2023
strchr()	2024
strcmp()	2025
strcoll()	2027
strcoll_l()	2027
strcpy()	2029
strcspn()	2032
strdup()	2033
stream	93
byte-oriented	498
interaction with file descriptors	497
stream orientation	498
wide-oriented	498
stream-full-policy attribute	539-540, 542, 1495
stream-min-size attribute	542, 1498
STREAMS	25, 93, 236, 366, 482, 688, 807, 831, 1053, 1146, 1163, 1410, 1433, 1553, 1753
access	1772, 2312, 3586
end	501
head	93
head/tail	93
multiplexed	500
multiplexor	1154
overview	94
STREAM_MAX	272, 835, 893, 1437, 2100, 2162, 3784
strerror()	2035
strerror_l()	2035
strerror_r()	2035
strfdinsert	366
strfmon()	2039
strfmon_l()	2039
strftime()	2044
strftime_l()	2044
string	94
strings	3259, 3774

striectl	366
strip	3262, 3774
strlen()	2053
strncasecmp()	2021, 2055
strncasecmp_l()	2021, 2055
strncat()	2056
strncmp()	2057
strncpy()	2058
strndup()	2033, 2060
strnlen()	2061
strpbrk()	2062
strpeek	366
strptime()	2063
strchr()	2068
strrecvfd	366
strsignal()	2069
strspn()	2070
strstr()	2071
strtod()	2072
strtof()	2072
strtoimax()	2076
strtok()	2078
strtok_r()	2078
strtol()	2081
strtold()	2072, 2084
strtoll()	2081, 2085
strtoul()	2086
strtoull()	2086
strtoimax()	2076, 2089
structures, additions to	3568
strxfrm()	2090
strxfrm_l()	2090
str_	475
str_list	366
str_mlist	367
stty	3264, 3538, 3774
combination modes	3269
control modes	3264
input modes	3265
local modes	3267
output modes	3266
special control character assignments	3268
ST_	476
st_	476
st_gid	2467
st_mode	2467
st_mtime	2467
ST_NOSUID	397, 785, 971
ST_RDONLY	397, 971
st_size	2467
st_uid	2467

su	
rationale for omission.....	3759
subpro ling.....	21,3486
subprofiling option groups.....	3789
subshell.....	94, 3496
successfully completed.....	3508
successfully transferred.....	94
sum.....	3716
rationale for omission.....	3759
sun_.....	476
superuser.....	572, 672, 1246, 2200, 2784, 2930, 3095, 3488, 3502, 3511, 3727, 3757
supplementary group ID.....	94, 2327, 3502
supplementary groups.....	672, 1042, 3511
supported threads functions.....	3640
suseconds_t.....	402
suspended job.....	94
SVID.....	1982
SVR4.....	1342, 1389
sv_.....	475
SV_.....	477
swab().....	2092
swapcontext().....	3692
swprintf().....	993, 2093
swscanf().....	1003, 2094
SWTCH.....	477
symbolic constant.....	95, 3482, 3503
symbolic link.....	95, 3503
symbolic name.....	3482
symbols.....	3566
POSIX.1.....	472
symlink().....	2095
symlinkat().....	2095
SYMLINK_MAX.....	274, 282, 902, 2096
SYMLOOP_MAX.....	272, 2100, 3573
SYMTYPE.....	413
sync().....	2098
synchronized I/O.....	3595, 3769
completion.....	95
data integrity completion.....	95, 3508, 3595
file integrity completion.....	96, 3508, 3595
synchronized I/O operation.....	96
synchronized input and output.....	95
synchronous I/O operation.....	96
synchronously accept a signal.....	1987
synchronously-generated signal.....	96, 3508
sysconf().....	2099 , 3493, 3598, 3601-3602, 3650, 3709, 3767, 3769
syslog().....	694, 2106, 3774
system.....	96
boot.....	96
call.....	3508
clock.....	96
configuration values.....	2831

console	97, 3508
crash	97, 974
database	3508
databases	97
documentation	97
name	2191, 3334
process	97, 3508
reboot	97, 3508
trace event	97
trace event type definitions	542
system documentation	3481
system environment	3766, 3774
System III	672, 2191, 3500, 3689
system interfaces	551, 3691
System V	556, 595, 672, 793, 826, 904, 1069, 1227, 1317, 1826, 1922, 1955, 1982, 2124
.....	2191, 3491, 3497, 3576
system()	2107, 3771, 3774-3775
system-wide	98
S_	475
s_	475
S_	477
S_ constants	
defined in <sys/stat.h>	393
S_ macros	
defined in <sys/stat.h>	393
S_BANDURG	368, 1148
S_ERROR	368, 1148
S_HANGUP	368, 1148
S_HIPRI	368, 1147
S_IFBLK	393, 1326
S_IFCHR	393, 1326
S_IFDIR	393, 1326
S_IFIFO	393, 1326
S_IFLNK	393
S_IFMT	393
S_IFREG	393, 1326
S_IFSOCK	393
S_INPUT	368, 1147
S_IRGRP	811, 962, 965, 1326
S_IROTH	811, 962, 965, 1326
S_IRUSR	811, 962, 965, 1326
S_IRWXG	1326
S_IRWXO	1326
S_IRWXU	1326
S_ISBLK	393
S_ISCHR	393
S_ISDIR	393
S_ISFIFO	393
S_ISGID	395, 665, 667-668, 1326, 2178, 2311
S_ISLNK	394
S_ISREG	393
S_ISSOCK	394

S_ISUID.....	395, 665, 667, 1326, 2178, 2311
S_ISVTX.....	665, 1326
S_IWGRP.....	811, 962, 965, 1326
S_IWOTH.....	811, 962, 965, 1326
S_IWUSR.....	811, 962, 965, 1326
S_IXGRP.....	1326
S_IXOTH.....	1326
S_IXUSR.....	1326
S_MSG.....	368, 1147
S_OUTPUT.....	368, 1147
S_RDBAND.....	368, 1147-1148
S_RDNORM.....	368, 1147
S_TYPEISMQ.....	394
S_TYPEISSEM.....	394
S_TYPEISSHM.....	394
S_TYPEISTMO.....	394
S_WRBAND.....	368, 1147
S_WRNORM.....	368, 1147
tab character.....	98
TABDLY.....	417
TABn.....	417
tabs.....	3273 , 3772
TABSIZE.....	629, 1290
tag file creation.....	2621
tail.....	3277 , 3772
talk.....	3281 , 3773
tan().....	2112
tanf().....	2112
tanh().....	2115
tanhf().....	2115
tanh1().....	2115
tanl().....	2112, 2117
tar	
rationale for omission.....	3759
tar format.....	3086
target queue.....	2445
target rule.....	2969
tcdrain().....	2118
tcow().....	2120
tcush().....	2122
tcgetattr().....	2124 , 3495
tcgetpgrp().....	2126 , 3496, 3550
tcgetsid().....	2128
TCIFLUSH.....	419, 2122
TCIOFF.....	419, 2120
TCIOFLUSH.....	419, 2122
TCION.....	419, 2120
TCOFLUSH.....	2122
TCOOFF.....	419, 2120
TCOON.....	419, 2120
TCP_.....	475
TCP_NODELAY.....	311

TCSADRAIN.....	419, 2131
TCSAFLUSH.....	419, 2131
TCSANOW.....	419, 2131
tcsendbreak().....	2129
tcsetattr().....	2131 , 3495, 3548
tcsetpgrp().....	2134 , 3495-3496
TCT.....	10
tdelete().....	2136
tee.....	3285 , 3771
TEF.....	11
telldir().....	2141
tempnam().....	2142
TERM.....	178 , 3538
terminal.....	98
access control.....	2124, 2132, 3551
controlling.....	200
device file.....	3549
device file, closing.....	3553
device name.....	2182
type.....	198, 3548
terminal characteristics	
stty.....	3264
tabs.....	3273
tput.....	3307
tty.....	3322
terminal device.....	98
terminate a process.....	555, 2879
terminology.....	3480
termios.....	199
canonical mode input processing.....	202
control modes.....	209
controlling terminal.....	200
input modes.....	206
local modes.....	210
non-canonical mode input processing.....	202
output modes.....	207
process group.....	200
special control characters.....	212
termios structure.....	2124, 3553
test.....	3288 , 3771, 3773
TeX.....	3773
text column.....	98
text file.....	98, 3509
tfind().....	2136, 2144
tgamma().....	2145
tgammaf().....	2145
tgamma1().....	2145
TGEXEC.....	413
TGREAD.....	413
TGWRITE.....	413
THOUSEP.....	267
thread.....	99, 3509

thread cancelability	
states.....	3657
type.....	3657
thread cancellation.....	3655, 3657
cleanup handlers.....	520
thread cancellation points.....	3657
thread concurrency level.....	3645
thread creation.....	1634
thread creation attributes.....	1563, 3633
thread ID.....	99, 513, 1638, 3509, 3650
thread interactions.....	3661
thread list.....	99
thread mutex.....	514, 3651
thread read-write lock.....	3659
thread scheduling.....	515, 3651
thread stack guard size.....	3646
thread termination.....	1639
thread-safe function.....	3509
thread-safety.....	99, 114, 513, 875, 3509, 3520, 3647
rationale.....	3520
thread-specific data.....	3635
thread-specific data key.....	99
creation.....	1653
deletion.....	1655
thread-specific data management.....	1647
threads.....	512, 3633
extensions.....	3642
implementation models.....	3635
regular file operations.....	522
tilde.....	99
tilde expansion.....	2354, 3728
time.....	3297 , 3771, 3774
time().....	2148 , 3572
timeouts.....	100, 3626
timer.....	100
ID.....	2153
overrun.....	100
timers.....	3617
TIMER_.....	476
timer_.....	476
TIMER_ABSTIME.....	426, 511, 683, 2155, 3618-3619
timer_create().....	2151
timer_delete().....	2154
timer_getoverrun().....	2155
timer_gettime().....	2155
TIMER_MAX.....	272, 2100, 3784
timer_settime().....	2155, 3618-3619
timer_t.....	402
times.....	2418
times().....	2158 , 3572, 3625, 3768
timespec.....	425 , 3519
timestamp clock.....	3687

timeval	380, 399
timezone().....	2161
time_t	402, 3519
tm.....	425
TMAGIC	413
TMAGLEN.....	413
TMPDIR.....	178, 3072
tmp le().....	2162
tmpnam()	2164
TMP_MAX.....	355, 2142, 2163-2164
tms	401
tms_.....	476
tm_.....	476
toascii().....	2166
TOEXEC.....	413
token.....	100
token recognition.....	2347, 3720
tolower().....	2167
tolower_l()	2167
TOREAD.....	413
TOSTOP	418, 817, 859, 923, 927, 958, 960, 2313, 3495
touch.....	3301, 3716, 3773
toupper().....	2169
toupper_l().....	2169
towctrans().....	2171
towctrans_l().....	2171
towlower()	2173
towlower_l()	2173
TOWRITE	413
towupper().....	2175
towupper_l().....	2175
TPL.....	11
TPP	11
TPS.....	11
tput	3307, 3774
tr.	3310, 3772-3773
trace analyzer.....	3676
trace analyzer process.....	100
trace controller process.....	100
trace event	100
POSIX_TRACE_ERROR.....	543
POSIX_TRACE_FILTER.....	543, 1529
POSIX_TRACE_OVERFLOW	543
POSIX_TRACE_RESUME.....	543
POSIX_TRACE_START	543, 1538
POSIX_TRACE_STOP	543, 1538
trace event type	100, 3685
filtering	3685
mapping.....	101
trace examples	3674
trace filter.....	101
trace functions	546

trace generation version.....	101
trace log	101
trace model.....	3669
trace operation control	3674
trace point.....	101
trace storage	3673
trace stream	101
attribute	3678
identifier	101
states.....	3672
trace system.....	101
trace-name attribute.....	542, 1492
traced process	102
TRACE_EVENT_NAME_MAX	273, 1517, 1519
TRACE_NAME_MAX.....	273
TRACE_SYS_MAX.....	273, 1514
TRACE_USER_EVENT_MAX.....	273, 1517, 1519
tracing	25, 115
TRACING	1490, 1492, 1494, 1497, 1509, 1511, 1513, 1517, 1519, 1522, 1524, 1527, 1529
.....	1532, 1538
tracing	3520, 3664
all processes	3672
detailed objectives.....	3665
status of a trace stream	102
track batch job request.....	2449
trap	2420 , 3747
TRAP	475
TRAP_BRKPT	337
TRAP_TRACE	337
TRC.....	11
TRI	11
triggering.....	3687
TRL	11
troff.....	3773
trojan horse.....	2931, 3488
true	2344, 2367, 3317, 3717, 3771
trunc().....	2177
truncate().....	2178
truncation-status attribute	1517
truncf().....	2177, 2180
truncl().....	2177, 2180
TSA	12
tsearch().....	2136, 2181
TSGID.....	413
TSH.....	12
tsort.....	3319
TSP.....	12
TSS.....	12
TSUID	413
TSVTX	413
tty.....	3322 , 3774
ttyname().....	2182 , 3647

ttynam_r()	2182
TTY_NAME_MAX	273, 2100, 2182, 3784
TUEXEC	413
TUREAD	413
TUWRITE	413
TVERSION	413
TVERSLEN	413
tv_	476
twalk()	2136, 2184
TYM	12
type	2344, 2367, 3324
typed memory	3605
name space	102
object	102
pool	102
port	102
TZ	179 , 3539
tzname	2185
TZNAME_MAX	273, 2100, 3784
tzset	2185
tzset()	2185 , 3771
T_FMT	267
T_FMT_AMPM	267
t_scalar_t	366
t_uscalar_t	366, 1149
ualarm()	3694, 3768
UCHAR_MAX	280-281
ucontext_t	335
uc_	475
UID_MAX	3690
uid_t	402, 3510
UINT	477
UINTMAX_MAX	352
UINTN_MAX	351
UINTPTR_MAX	352
UINT_FASTN_MAX	351
UINT_LEASTN_MAX	351
UINT_MAX	281, 595, 1999
UIO_MAXIOV	476
ulimit	2344, 2367, 3326
ulimit()	2187
ULLONG_MAX	281, 2087
ULONG_MAX	281, 2087, 2285, 3709
UL_	476
UL_GETFSIZE	433, 2187
UL_SETFSIZE	433, 2187
umask	2344, 2367, 3328, 3717, 3774
umask()	2189 , 3768
umount()	3498
unalias	2344, 2367, 3332, 3717, 3772
uname	3334 , 3774
uname()	2191 , 3767

unary primaries	3290
unbind.....	102
unbounded priority inversion.....	3655
uncompress	3337
unde ned	63481
rationale.....	3481
underlying function	498
unexpand.....	3340, 3772
unget	3343
ungetc()	2193
ungetwc()	2195
unicast.....	532
uniq.....	3346, 3772-3773
unlink	3351
unlink()	2197, 3574, 3601, 3603-3604, 3769
unlinkat()	2197
unlockpt()	2203
unpack	
rationale for omission.....	3759
unsafe functions	3583
unset.....	2424
unsetenv().....	2204
unspeci ed	63481
rationale.....	3481
until loop	2374, 3744
UP	12
upper multiplexing.....	94
upshifting	103
US-ASCII.....	1162
uselocale().....	2205
user database	103, 3510
access.....	3510
user ID.....	103, 2857
logname	2916
newgrp.....	3023
real and effective	1918
setting real and effective	1918
who.....	3440
user name	103
user requirements.....	3763
user trace event.....	103
user trace event type definitions.....	545
User_List.....	2440
USER_PROCESS	456, 771-772
USHRT_MAX	281
usleep().....	3694, 3768
ustar format.....	3086
UTC	2185
utility	104, 119, 3520
argument syntax.....	3554
conventions	3554
description defaults	3712

limits.....	3709
option parsing.....	2837
syntax guidelines.....	216, 3555
utimbuf.....	455
utime().....	2207 , 3768
utimensat().....	987, 2210
utimes().....	987, 2210
UTIME_NOW.....	394, 987
UTIME_OMIT.....	394, 987
utim_.....	476
utmpx.....	456
uts_.....	476
ut_.....	476
UU.....	12
uucp.....	3353 , 3773
uudecode.....	3357 , 3772-3773
uuencode.....	3360 , 3772-3773
uustat.....	3365
uux.....	3368
val.....	3372
variable.....	104, 3721
variable assignment.....	119, 3520
Variable_List.....	2440
va_arg().....	2211
va_copy().....	2211
va_end().....	2211
va_start().....	2211
VDISCARD.....	477
vdprintf().....	2212
VDSUSP.....	477
VEOF.....	415, 3554
VEOL.....	415, 3554
VERASE.....	415
Version 7.....	595, 1227, 2191, 3517, 3718
vertical-tab character.....	104
vfork().....	3694
vfprintf().....	2212
VFS.....	397
vfscanf().....	2214
vfwprintf().....	2215
vfwscanf().....	2217
vhangup().....	3497
vi.....	3375 , 3772-3773
<ESC>.....	3415
append.....	3396
change.....	3397
change to end-of-line.....	3398
clear and redisplay.....	3383
command descriptions.....	3376
control-D.....	3412
control-H.....	3412
control-T.....	3414

control-U.....	3414
control-V.....	3414
current and line above.....	3390
delete.....	3398
delete character.....	3408
delete to end-of-line.....	3399
display information.....	3382
edit the alternate file.....	3384
enter ex mode.....	3404
execute.....	3395
execute an ex command.....	3394
exit.....	3410
find character.....	3399-3400
find regular expression.....	3392
Initialization.....	3376
input mode commands.....	3410
insert.....	3401
insert empty line.....	3403
join.....	3401
mark position.....	3402
move back.....	3390-3391, 3396-3397
move cursor.....	3382, 3385-3386, 3405-3406
move down.....	3383
move forward.....	3390-3391
move to bigword.....	3399, 3407
move to bottom of screen.....	3401
move to first character in line.....	3394
move to first non-<blank>.....	3390
move to line.....	3400
move to matching character.....	3387
move to middle of screen.....	3402
move to next section.....	3389
move to specific column.....	3391
move to top of screen.....	3400
move to word.....	3399, 3407
move up.....	3383
newline.....	3413
nul.....	3412, 3415
page backwards.....	3381
page forward.....	3382
put from buffer.....	3403-3404
redraw screen.....	3384
redraw window.....	3409
regular expression.....	3395
repeat.....	3392
repeat find.....	3394, 3402
repeat substitution.....	3388
replace character.....	3404-3405
replace text with command.....	3386
return to previous context.....	3388
return to previous section.....	3389
reverse case.....	3396

reverse find character	3391
scroll backward.....	3384
scroll backward by line.....	3384
scroll forward.....	3381
scroll forward by line.....	3381
search for tagstring	3385
shift left	3394
shift right	3395
substitute character	3405
substitute lines.....	3405
terminate command or input mode	3385
undo	3406
undo current line.....	3407
yank.....	3408
yank current line	3409
VINTR.....	415
virtual processor.....	3510
VISIT	2136, 2184
visual mode.....	2697
VKILL.....	415
VLNEXT	477
VMIN	3554
vprintf().....	2212, 2218
VQUIT.....	415
VREPRINT	477
vscanf().....	2214, 2219
vsprintf().....	2212, 2220
vsprintf()	2212, 2220
vsscanf()	2214, 2221
VSTART	415
VSTATUS.....	477
VSTOP.....	415
VSUSP	415
vswprintf().....	2215, 2222
vswscanf().....	2217, 2223
VTDLY	417
VTIME.....	3554
VTn.....	417
VWERASE.....	477
vwprintf().....	2215, 2224
vwscanf().....	2217, 2225
wait.....	2344, 2367, 3430, 3717, 3771
for process termination	2232
for thread termination	1650
wait()	2226, 3572, 3576, 3581, 3583, 3585, 3768
waitid().....	2236, 3583, 3691, 3768
waiting on a condition.....	1619
waitpid()	2226, 2239, 3495, 3499, 3583, 3689, 3768
wall	
rationale for omission.....	3759
WARNING	888

warning	
OB	9
OF	9
wc	3434 , 3772
WCHAR_MAX	352, 458
WCHAR_MIN	352, 458
WCONTINUED	409, 2226, 2236
wcpcpy()	2240 , 2251
wcpncpy()	2241 , 2260
wcrtomb()	2242
wscasecmp()	2244
wscasecmp_l()	2244
wscat()	2246
wcschr()	2247
wscmp()	2248
wscoll()	2249
wscoll_l()	2249
wscopy()	2251
wscspn()	2252
wcsdup()	2253
wcsftime()	2254
wcslen()	2256
wcsncasecmp()	2244, 2257
wcsncasecmp_l()	2244, 2257
wcsncat()	2258
wcsncmp()	2259
wcsncpy()	2260
wcsnlen()	2256, 2262
wcsnrtombs()	2263 , 2266
wcspbrk()	2264
wcsrchr()	2265
wcsrtombs()	2266
wcsspn()	2268
wcsstr()	2269
wcstod()	2270
wcstof()	2270
wcstoimax()	2274
wcstok()	2275
wcstol()	2277
wcstold()	2270, 2280
wcstoll()	2277, 2281
wcstombs()	2282
wcstoul()	2284
wcstoull()	2284
wcstoumax()	2274, 2287
wcswcs()	3694
wcswidth()	2288
wcsxfrm()	2289
wcsxfrm_l()	2289
wctob()	2291
wctomb()	2292
wctrans()	2294

wctrans_l()	2294
wctype()	2296
wctype_l()	2296
wcwidth()	2298
WEOF	458, 463, 548, 1194, 1196, 1200, 1202, 1205, 1207, 1209, 1211, 1213, 1215, 1217, 1219, 2173, 2175, 2195
WERASE	3551
WEXITED	409, 2236
WEXITSTATUS	359, 409, 2227
we_	476
what	3437
while loop	2373, 3744
white space	104
who	3440 , 3773-3774
wide characters	128
wide-character code	3522
wide-character code (C language)	104
wide-character input/output functions	105
wide-character string	105
wide-oriented stream	498
WIFCONTINUED	409, 2227
WIFEXITED	359, 409, 2227
WIFSIGNALED	359, 409, 2227
WIFSTOPPED	359, 409, 2227, 2233
WINT_MAX	352
WINT_MIN	352
wmemchr()	2299
wmemcmp()	2300
wmemcpy()	2301
wmemmove()	2302
wmemset()	2303
WNOHANG	359, 409, 1955, 2226, 2236
WNOWAIT	409, 2236, 3691
word	105
word counting	3434
word expansions	2353, 3727
wordexp()	2304 , 3774
wordfree()	2304, 3774
WORD_BIT	280-281
working directory	105
worldwide portability interface	105
wprintf()	993, 2309
WRDE_	476
WRDE_APPEND	465, 2305
WRDE_BADCHAR	465, 2306
WRDE_BADVAL	465, 2306
WRDE_CMDSUB	465, 2306
WRDE_DOOFFS	465, 2305
WRDE_NOCMD	465, 2305
WRDE_NOSPACE	465, 2306
WRDE_REUSE	465, 2305
WRDE_SHOWERR	465, 2305

WRDE_SYNTAX	465, 2306
WRDE_UNDEF	465, 2305
write	105, 3444, 3772-3773
write lock	3644
write to a file	2314
write()	2310 , 3494-3495, 3551, 3572, 3581-3582, 3585, 3595-3596, 3600-3601 3690, 3769
writev()	2319
writing data	3552
wscanf()	1003, 2321
WSTOPPED	409, 2236
WSTOPSIG	359, 409, 2227
WTERMSIG	359, 409, 2227
WUNTRACED	359, 409, 2227, 2232, 3495
W_OK	441
xargs	3447 , 3771
XOPEN_UNIX	19, 29
XOPEN_UUCP	19, 29
XSI	12 , 105, 3511
conformance	15, 19, 106
XSI interprocess communication	501
XSI IPC	3586
XSI options groups	22
XSI STREAMS	25
XSI system interfaces	
conformance	19
XSI_C_LANG_SUPPORT	3793
XSI_DBM	3794
XSI_DEVICE_IO	3794
XSI_DEVICE_SPECIFIC	3794
XSI_FILE_SYSTEM	3794
XSI_IPC	3794
XSI_JUMP	3794
XSI_MATH	3794
XSI_MULTI_PROCESS	3794
XSI_SIGNALS	3794
XSI_SINGLE_PROCESS	3794
XSI_SYSTEM_DATABASE	3794
XSI_SYSTEM_LOGGING	3794
XSI_THREADS_EXT	3794
XSI_TIMERS	3794
XSI_USER_GROUPS	3794
XSI_WIDE_CHAR	3794
XSR	13
X_OK	441, 573
y0()	2322
y1()	2322
yacc	3454 , 3774, 3776
algorithms	3465
code file	3456
completing the program	3465
conflicts	3463

debugging the parser.....	3465
declarations section.....	3457
description file	3456
error handling.....	3463
grammar rules	3459
header file.....	3456
input grammar.....	3461
input language.....	3456
interface to the lexical analyzer.....	3464
lexical structure of the grammar	3457
library.....	3465
limits.....	3466
programs section	3460
YESEXPR	267
yn().....	2322
zcat	3471
zombie process	106

Consensus

WE BUILD IT.

Connect with us on:



Facebook: <https://www.facebook.com/ieeesa>



Twitter: @ieeesa



LinkedIn: <http://www.linkedin.com/groups/IEEESA-Official-IEEE-Standards-Association-1791118>



IEEE-SA Standards Insight blog: <http://standardsinsight.com>



YouTube: IEEE-SA Channel